Wind-US Installation Guide*

The NPARC Alliance

NASA Glenn Research Center
Cleveland, Ohio

USAF Arnold Engineering Development Complex
Tullahoma, Tennessee

*This is an unnumbered version of this document, created January 19, 2024. Please send corrections, additions, ideas, etc., to Dennis Yoder at *Dennis.A.Yoder@nasa.gov*.

Contents

1 Overview

The NPARC Alliance flow simulation system is distributed to users as a collection of compressed tar files containing the source code to Wind-US and several pre- and post-processing tools. These build distributions enable users to compile optimized executables for their particular computational platform(s).

Wind-US Build Distribution

The Wind-US build distribution contains all the files needed to build, install, and run Wind-US on a variety of platforms. This includes:

- Source code for the Wind-US flow solver.
- Source code for all the required library routines, including PVM but not MPI.
- Scripts needed to run Wind-US.
- Makefiles for a variety of computational platforms.

This distribution includes the PVM source code but not MPI, since there are several popular versions of MPI. If Wind-US is to be run on a multi-processor system using MPI, you will need to configure the makefiles to compile and link with your locally-installed MPI libraries.

Tools Build Distribution

The tools build distribution contains all the files needed to build, install, and run one or more of the tools on a variety of platforms. These tools can briefly be summarized as:

- GMAN, a pre-processor used for setting boundary conditions and multi-zone connectivity. (see the *GMAN User's Guide*)
- MADCAP, intended primarily as a successor to GMAN, and potentially as a single user interface for accessing a range of tools and processes used to perform CFD analyses. (see the *MADCAP User's Guide*)
- CFPOST, a post-processor used to list and plot results, generate reports, and produce files for other post-processors. (see the *CFPOST User's Guide*)
- A variety of smaller utilities, described in the *Wind-US Utilities Guide*.

The tools build distribution includes:

- Source code for the tools
- Source code for all the required library routines
- Scripts needed to run the tools
- Makefiles for a variety of computational platforms

Process Build Distributions

The process build distributions contain the files needed to build other (typically third-party) software that supports the Wind-US simulation process. This includes:

- HTX heat transfer solver
- FieldView Reader
- Tecplot Loader

These distributions come with their own installation instructions and may require you to first compile the shared libraries that are part of the Wind-US build distribution.

Note that the availability of this software does not constitute an official endorsement by the NPARC Alliance or its members. Other vendors and developers that would like to include Wind-US compatibility in their products are encouraged to contact us via the support web page.

2 Obtaining Wind-US and the Tools

The Wind-US flow solver and all related utilities are no longer under active development and are no longer being distributed to users outside of NASA.

NASA users may request a copy via the support web page at: https://www.grc.nasa.gov/www/winddocs/support/index.html

Approved users will receive build distributions in the form of a compressed tar file with a *.tgz* extension. As described in Section 1, these build distributions enable users to compile optimized executables for their particular computational platform(s).

2.1 Wind-US Build Distribution

The Wind-US build distribution contains all the files needed to build, install, and run Wind-US on a variety of platforms.

2.2 Tools Build Distribution

The tools build distribution contains all the files needed to build, install, and run one or more of the various pre- and post-processing tools on a variety of platforms.

2.3 Process Build Distribution

The process build distributions contain the files needed to build other (typically third-party) software that supports the Wind-US simulation process.

3 Installing the Build Distributions

Wind-US and the tools are designed to be compiled in a build (or development) directory and then the final executables installed into an application directory. This allows one to install multiple versions of Wind-US into a single application directory that uses shared run-time scripts. The user can select which version to use at run-time. The following sections describe the steps necessary to install the build distributions in this manner.

3.1 Building Wind-US

Makefiles are distributed with the Wind-US build distribution for a variety of platform, operating system, and CPU combinations, including:

- Convex
- Cray
- Hewlett-Packard
  - HP-UX 11.*x*; PA-8600 processor
- IBM RS6000; PC600 processor
- Linux
  - *glibc* 2.3-2.19; Athlon, Xeon, Opteron, and X86 processors
  - SUSE; Opteron processor
- Silicon Graphics IRIX 6.5; R4400, R5000, R8000, R10000, R12000, R14000, R16000 processors
- Sun Solaris 8 and 9; SPARC4U processor

If you need to build the code for a platform not listed above, you will need to first create the appropriate makefiles, using the existing ones as a starting point. See Section 5 for further information.

Assuming the appropriate makefiles exist, the recommended procedure to build and install Wind-US from source is as follows. *Please re-read each step completely before executing any commands that are described.*

1. Create an appropriate top-level directory.

   ```
   mkdir -p $HOME/WINDUS
   cd       $HOME/WINDUS
   ```

   This will ultimately contain the build/development source (*wind-dev*) and application executable (*wind*) subdirectories.

   If an existing top-level Wind-US installation directory is to be used, the existing *wind-dev* subdirectory should be renamed so that it is not overwritten. Any makefiles you may have modified within this directory may be of use in constructing makefiles for the new installation.

   When installing multiple versions into a shared installation directory, one should start with the oldest version and work towards the newest. This way the most current run scripts will be made available. Installing an older version into an existing directory might install older run scripts that do not support more recent code options.

2. Put the gzip'ed tar file containing the Wind-US build distribution into this directory. You should now have a file of the form:

   dist.windus.x.xxx.tgz

3. Unpack the Wind-US build distribution by doing:

   `gunzip -c` *filename* `| tar xvf -`

where *filename* is the file name, including the *.tgz* extension. This creates a directory named *wind-dev* containing the Wind-US source code. There will also be two scripts to assist with the installation: *Install.appl* and *Install.tools*.

If you also plan to install the tools, you should follow those instructions in parallel with these. In particular, it is important to unpack the tools build bundle before proceeding with any changes to the makefiles. Otherwise those changes could be overwritten when you do eventually unpack the tools.

4. Use the installation script to create the application directory and copy some necessary scripts. Responses to the interactive prompts are shown in italic font.

```
./Install.appl

Under what parent directory should Wind-US be installed?
  (Default is current directory):
```
Hit enter to accept the default directory.

```
Run scripts will be installed in $HOME/WINDUS/wind/bin
Is this OK? (y/n [y])
```
Answer: y

```
Copying files to $HOME/WINDUS/wind
Modifying cfd.login and cfd.profile

Wind-US scripts have been copied to the installation directory.
To finish the installation ...

csh/tcsh users: add the following line to your .login file
   source $HOME/WINDUS/wind/bin/cfd.login

sh/bash/ksh users: add the following line to your .profile file
   . $HOME/WINDUS/wind/bin/cfd.profile
```

5. Update the login scripts

   Follow the instructions described by the script output to update your login scripts. At the same time, define the environment variable `WIND_DEV` as the location of the directory containing the top-level Makefile. This should be the full path to the *wind-dev* directory.

   Typically, *csh* and *tcsh* users must edit their *$HOME/.login* file and add the following two lines:

```
setenv WIND_DEV "path_name/wind-dev"
source path_name/wind/bin/cfd.login
```

   where again *path_name* is the full path name of the directory used in step 1. Similarly, *bash*, *ksh*, and *sh* users must edit their *$HOME/.profile* file and add the following two lines:

```
WIND_DEV="path_name/wind-dev"; export WIND_DEV
. path_name/wind/bin/cfd.profile
```

   This causes the contents of the *cfd.login* (or *cfd.profile*) file to be executed automatically at login time to set up the environment variables `CFDROOT`, `SYSTEM`, and `SYSTEM_CPU`, that are required for installing and running Wind-US, and to modify your `PATH` to include the location of the Wind-US executable.

6. At this point, log out and log back in to execute your *.login* or *.profile* file (depending on your login shell). Alternatively, in your home directory execute (for *csh* and *tcsh* users)

   ```
   source .login
   ```

   or (for *sh* and *ksh* users)

   ```
   . .profile
   ```

7. Check to see if the environment variables have been set to appropriate values, by doing

   ```
   printenv WIND_DEV
   printenv CFDROOT
   printenv SYSTEM
   printenv SYSTEM_CPU
   ```

   `WIND_DEV` was defined in step 5. `CFDROOT` should have a value of *path_name/wind* where *path_name* is the full path name of the directory used in step 1. This represents the root location that contains the Wind-US executable(s) and scripts. `SYSTEM` and `SYSTEM_CPU` should describe the operating system and cpu type. If all are correctly set, skip the next step.

8. If `CFDROOT` is not set properly, you will need to manually edit the Wind-US login script. Denoting *path_name* as the full path name (starting with a "/") of the directory used in step 1, users of the *csh* or *tcsh* shell should edit the file *path_name/wind/bin/cfd.login* to modify the definition of `CFDROOT` as follows:

   ```
   if (! "$?CFDROOT") then
   #=BEGROOT=
     setenv CFDROOT "path_name/wind"
   #=ENDROOT=
   endif
   ```

   Similarly, *bash*, *ksh*, or *sh* users would edit the file *path_name/wind-dev/bin/cfd.profile* to modify the definition of `CFDROOT` to:

   ```
   if [ ! "$CFDROOT" ] ; then
   #=BEGROOT=
     CFDROOT="path_name/wind"
   #=ENDROOT=
     export CFDROOT
   fi
   ```

   Return to step 6 to have this change take effect.

9. Now that the environment variables are set properly, move into the `WIND_DEV` directory.

   ```
   cd $WIND_DEV
   ```

10. Configure the makefiles.

    The default values in the configuration files should be sufficient to produce an executable on most well-configured systems (assuming that compilers are installed). The primary exception to this is Linux, which does not have a "default" Fortran 90/95 compiler. Another potential problem is the location of the MPI libraries, which needs to be specified if you wish to use MPI for parallel runs. To be safe, review the contents of the following files or parts of files, where *SYSTEM* and *SYSTEM_CPU* correspond to the `SYSTEM` and `SYSTEM_CPU` environment variables, paying special attention to the items noted below.

When modifying files it is always a good idea to save a copy of the original (i.e., as **.bak*), and for new files to create an empty **.bak* file. This makes it easy to identify which files you have changed, and you can use a visual difference program to compare those changes against the original.

- *Makefile.configure*
  - If you are compiling on a multi-processor system with MPI message passing available, change the definition of `USE_MPI` from `NO` to `YES`. This will link in the MPI libraries, allowing the use of MPI message passing for parallel processing. Note, though, that because Wind-US uses dynamically-linked libraries, an executable created with `USE_MPI` equal to `YES` will not run on multi-processor systems without MPI.
  - If you will never be running parallel jobs using PVM message passing, change the definition of `USE_PVM` from `YES` to `NO`.
  - The value of `PRECISION` may be changed to make all Fortran variables single or double precision. The default is a mix, with single precision for the mean flow solution and boundary conditions, and double precision for chemistry data, right-hand-side data, grid coordinates, and grid metrics.
- *source/Makefile.user*
  - Defines which of the Wind-US modules should be compiled. Only in rare circumstances does this file need to be modified.
- *source/makefiles/Makefile.include.SYSTEM.SYSTEM_CPU.xxx*, where *xxx* corresponds to *opt*, *dbx*, *pure*, or *check*. Most users compile the optimized (*opt*) version.
  - This file contains the compiler optimizations for the specific system being used.
  - Check the definition of `CPP`, which is the full path name for the C pre-processor *cpp*, to make sure it is correct for your system. On many systems, you can locate *cpp* using the *whereis* command.

    ```
    whereis -b cpp
    ```

  - For Linux systems, you will most likely have to change the definitions related to the Fortran and C compilers being used (i.e., the variables `ABI`, `FC`, `F90`, `LD`, etc.) These lines should be commented/uncommented as needed for your system, but additional changes may also be necessary or desired.
  - If you are compiling on a multi-processor system with MPI message passing, check the definition of `MPILIBS`, and modify it if necessary to use a different version or a non-standard location.
  - If your system doesn't support the Fortran long integer data type, add "`-DNOFLONG`" (without the quotes) to the definition of `WINDDEFS`. Errors in the compilation of *mem_management_module.f90* about an ambiguous definition for the generic interfaces `alloc` and `dealloc`, involving the specific interfaces for `allockindlong_1` and `allockindint_1`, etc., are an indication that "`-DNOFLONG`" is needed.

11. Compile the Wind-US source.

    From the `WIND_DEV` directory, run *make* to create all the required libraries and executables for running Wind-US. The output from the compilation process is very lengthy. To capture

the output from *make* in the file *make.log* for later examination if something goes wrong, in addition to displaying it at the terminal, *csh* and *tcsh* users should do

```
make opt |& tee make.log
```

and *sh* and *ksh* users should do

```
make opt 2>&1 | tee make.log
```

If the compilation is successfull, the Wind-US executable should be present in the directory *$WIND_DEV/SYSTEM/SYSTEM_CPU/bin*.

12. To install the compiled Wind-US and PVM executables into the application directory, use the following commands:

    ```
    make install
    make install_scripts
    make copy_pvm
    ```

    The first two commands copy the Wind-US executable to *$CFDROOT/SYSTEM/CPU/bin*, and the scripts to *$CFDROOT/bin*. The third copies the PVM executables and libraries to *$CFDROOT/pvm/lib/SYSTEM/CPU*, some PVM include files to *$CFDROOT/pvm/include*, and PVM scripts to *$CFDROOT/pvm/lib*. The parameters *SYSTEM* and *CPU* correspond to the `SYSTEM` and `SYSTEM_CPU` environment variables, respectively.

    Note that the Wind-US executables are named by major revision number (i.e., *Wind-US4.exe*). However, in some cases, such as before and after a bug fix, it is useful to retain minor revisions. To do so, before issuing the install commands listed above, you should rename the existing executable in the application directory (*$CFDROOT/SYSTEM/SYSTEM_CPU/bin*) to include the minor revision number (i.e., *Wind-US4.111.exe*). The *windver* script can be used to print the version number for a particular executable.

13. It would be best to log out and log back in before running Wind-US. This executes the shell start-up scripts, modifying the `PATH` environment variable to include the newly-created location for the Wind-US executable.

3.2 Building the Tools

The tools build distribution is designed to be a completely independent package, so that the tools can be built without requiring any additional files from Wind-US. Thus, one could have separate directory trees for the Wind-US build distribution and tools build distributions. This would lead to a great deal of duplication, however. Therefore, the build distributions are designed to overlay one another. The following instructions assume that all the tools are being built in the same directory tree. *Please re-read each step completely before executing any commands that are described.*

1. Create an appropriate top-level directory.

    ```
    mkdir -p $HOME/WINDUS
    cd       $HOME/WINDUS
    ```

    This will ultimately contain the build/development source (*wind-dev*), more specifically (*wind-dev/tools-dev*), and tools executable (*tools*) subdirectories.

    An existing top-level Wind-US directory is often used for this purpose, since it reduces the amount of extra disk space and configuration necessary. In this case, you will want to use the directory above *wind-dev*. See step 1 in Section 3.1.

2. Install lua version 4.01 if it is not already available on your system. Note that due to changes in the API, *newer versions of lua will not work with the Wind-US tools.* These instructions assume that lua will be installed under the top-level directory of step 1. However, it could be installed in its own stand-alone location.

   Download the source code from http://www.lua.org/ftp/lua-4.0.1.tar.gz and save as:

   lua.4.0.1.tar.gz

   Unpack the source files.

   `gunzip -c` *lua.4.0.1.tar.gz* `| tar xvf -`

   This will extract the *lua.4.0.1* directory. To compile lua,

   ```
   cd lua.4.0.1
   make
   ```

   This should create the following files:

   ```
   $HOME/WINDUS/lua-4.0.1/bin/lua
   $HOME/WINDUS/lua-4.0.1/bin/luac
   $HOME/WINDUS/lua-4.0.1/lib/liblua.a
   $HOME/WINDUS/lua-4.0.1/lib/liblualib.a
   ```

3. Install the CGNS library if it is not already available on your system. Once again, these instructions assume that it will be installed under the top-level directory of step 1. However, it could be installed in its own stand-alone location.

   Download cgnslib (2.5.5) from https://cgns.github.io/download.html and save as:

   cgnslib.2.5.5.tar.gz

   Newer versions of cgnslib may also work. Unpack the source files.

   `gunzip -c` *cgnslib.2.5.5.tar.gz* `| tar xvf -`

   This will extract the *cgns_2.5* directory. Compile the CGNS library with commands similar to the following. You may need to adjust the settings, depending on your system.

   ```
   cd $HOME/WINDUS/cgnslib_2.5
   ./configure --prefix=$HOME/WINDUS/cgnslib_2.5 --with-system=LINUX64 --enable-64bit
   make SYSTEM=LINUX64
   mkdir include
   mkdir lib
   make install
   ```

   This should create the following files:

   ```
   $HOME/WINDUS/cgnslib_2.5/include/cgnslib.h
   $HOME/WINDUS/cgnslib_2.5/include/cgnslib_f.h
   $HOME/WINDUS/cgnslib_2.5/include/cgnswin_f.h
   $HOME/WINDUS/cgnslib_2.5/lib/libcgns.a
   ```

4. Put the gzip'ed tar file containing the tools build distribution into the top-level directory used in step 1. You should now have a file of the form:

   dist.alltools.tgz

5. Unpack the tools build distribution by doing:

```
gunzip -c dist.alltools.tgz | tar xvf -
```

This extracts a directory named *wind-dev* containing the source code. The source code for individual tools will be located in subdirectories of *wind-dev/tools-dev*. There will also be two scripts to assist with the installation: *Install.appl* and *Install.tools*.

If you also plan to install Wind-US, you should follow those instructions in parallel with these. In particular, it is important to unpack all of the build bundles before proceeding with any changes to the makefiles. Otherwise those changes could be overwritten when unpacking another bundle.

6. Follow steps 4-8 in Section 3.1 and verify that the `WIND_DEV`, `CFDROOT`, `SYSTEM`, and `SYSTEM_CPU` environment variables are correctly defined.

7. Use the installation script to create the application directory and copy some necessary scripts. Responses to the interactive prompts are shown in italic font.

```
./Install.tools

Under what parent directory should the tools be installed?
  (Default is current directory):
Hit enter to accept the default directory.

Run scripts will be installed in $HOME/WINDUS/tools/bin
Is this OK? (y/n [y])
Answer: y

Copying files to $HOME/WINDUS/tools
Modifying tools.login and tools.profile

Tools scripts have been copied to the installation directory.
To finish the installation ...

csh/tcsh users: add the following line to your .login file
   source $HOME/WINDUS/tools/bin/tools.login

sh/bash/ksh users: add the following line to your .profile file
   . $HOME/WINDUS/tools/bin/tools.profile
```

8. Update the login scripts.

   Follow the instructions described by the script output to update your login scripts. Typically, *csh* and *tcsh* users must edit their *$HOME/.login* file and add the following line:

   ```
   source path_name/tools/bin/tools.login
   ```

   where again *path_name* is the full path name of the directory used in step 1. Similarly, *bash*, *ksh*, and *sh* users must edit their *$HOME/.profile* file and add the following line:

   ```
   . path_name/tools/bin/tools.profile
   ```

   This causes the contents of the *tools.login* (or *tools.profile*) file to be executed automatically at login time to set up the `TOOLSROOT` environment variable that is required for installing and running the tools, and to modify your `PATH` to include the location of the tools executables.

9. At this point, log out and log back in to execute your *.login* or *.profile* file (depending on your login shell). Alternatively, in your home directory execute (for *csh* and *tcsh* users)

```
source .login
```

or (for *sh* and *ksh* users)

```
. .profile
```

10. Verify that the `TOOLSROOT` environment variable is properly set, by doing:

    ```
    printenv TOOLSROOT
    ```

    It should have a value of *path_name/tools* where *path_name* is the full path name of the directory used in step 1. If it is, skip the next step.

11. If `TOOLSROOT` is not set properly, you may need to manually edit the tools login script. Denoting *path_name* as the full path name (starting with a "/") of the directory used in step 1, users of the *csh* or *tcsh* shell should edit the file *path_name/tools/bin/tools.login* to modify the definition of `TOOLSROOT` as follows:

    ```
    #=BEGROOT=
    setenv TOOLSROOT "path_name/tools"
    #=ENDROOT=
    ```

    Similarly, *bash*, *ksh*, or *sh* users would edit the file *path_name/tools/bin/tools.profile* to modify the definition of `TOOLSROOT` to:

    ```
    #=BEGROOT=
    TOOLSROOT="path_name/tools"
    #=ENDROOT=
    export TOOLSROOT
    ```

    Return to step 9 to have this change take effect.

12. Now that the environment variables are set properly, move into the `WIND_DEV` directory.

    ```
    cd $WIND_DEV
    ```

13. Configure the makefiles.

    Most systems other than Linux should be able to use the default values in the configuration files without problem. On Linux systems, it is likely that the definitions related to the Fortran and C compilers will have to be changed. To be safe, check the following files to be sure they contain information that is appropriate for your system. (See the notes for step 10 in Section 3.1.)

    - *Makefile.configure*
      - See the instructions in step 10 of Section 3.1
    - *source/makefiles/Makefile.include.SYSTEM.SYSTEM_CPU.opt*
      - See the instructions in step 10 of Section 3.1
      - Comment out (or not) the explicit static and shared library settings as appropriate:

        ```
        #TOOLS_STATLIB= -static
        #TOOLS_SHARLIB= -shared
        ```

        Note that the default behavior of the Intel compiler is to use shared libraries. In some versions of the compiler, specifying the *-shared* option leads to linker errors. Thus, for Intel compilers, it is recommended to leave both settings commented out.

  - Provide the proper locations of Tcl, Lua, and CGNS. These may differ depending on your system setup, but are typically set to:

```
TOOLS_TCLLIBS=  -L/usr/lib64 -ltcl8.5
TOOLS_LUALIBS=  -L/$(HOME)/WINDUS/lua-4.0.1/lib -llua -llualib
TOOLS_CGNSLIBS= -L/$(HOME)/WINDUS/cgnslib_2.5/lib -lcgns
TCL_INCLUDE=    $(INCCMD)/usr/include
LUA_INCLUDE=    $(INCCMD)/$(HOME)/WINDUS/lua-4.0.1/include
CGNS_INCLUDE=   $(INCCMD)/$(HOME)/WINDUS/cgnslib_2.5/include
```

  - Provide the proper locations of the compiler libraries defined by `TOOLS_OSLIBS`, `TOOLS_CPPLIBS`, and `MADCAP_OSLIBS`.

- *source/makefiles/pvm_conf/SYSTEM.SYSTEM_CPU.def.opt*
  - Do not bother configuring or creating this file unless you encounter an error during the compilation process. This file is used to override the default settings given in a similarly named *pvm/conf/SYSTEM.def* file.
- *tools-dev/Makefile*
  - This file defines the make dependencies and rarely needs to be changed.
- *tools-dev/libmdgl/SYSTEM.mkf*
  - This file is used to compile the graphics library needed by GMAN, CFPOST, and MADCAP.

14. Compile the tools source.

    In the `WIND_DEV` directory, run *make* to create all the required libraries and executables for all the tools for which you have installed source. To capture the output from *make* in the file *make_tools.log* for later examination if something goes wrong, in addition to displaying it at the terminal, *csh* and *tcsh* users should do

    ```
    make all_tools |& tee make_tools.log
    ```

    and *sh* and *ksh* users should do

    ```
    make all_tools 2>&1 | tee make_tools.log
    ```

    You can also compile tools individually, by doing

    `make` *tool_name*

    where *tool_name* is the name of the tool. Note that the names to be used for GMAN, CFPOST, and MADCAP are *gmanpre*, *cfpost_prod*, and *madcapprod*, respectively.

    The *$WIND_DEV/SYSTEM/SYSTEM_CPU/bin* directory will contain the executables for the various tools if the compilation is successful.

15. Copy the executable and library files from the build directory to the application directory where they will be used. This is done via the command:

    ```
    make install_tools
    ```

    This copies the tools executables to *$TOOLSROOT/SYSTEM/CPU/bin*, where *SYSTEM* and *CPU* correspond to the `SYSTEM` and `SYSTEM_CPU` environment variables, as described in step 11 in Section 3.1.

16. It would be best to log out and log back in again before running any of the tools. This executes the shell start-up scripts, modifying the `PATH` environment variable to include the newly-created location for the tools executables.

3.3 Tips for Installing on a UNIX System

3.3.1 Using Symbolic Links for Similar Systems

In general, it is recommended to compile executables on each machine type that you will use. However, in some situations two machines may be similar enough that the same executable can be used with only a minor degradation in performance. In that case, symbolic links may be used to specify that a different (but compatible) executable should be used instead.

Suppose you have compiled executables for a `LINUX64-GLIBC2.5` system with a `XEON` processor, but would also like to provide binaries for `OPTERON` processors. You can share the available executable by creating a symbolic link. The Wind-US, PVM, and tools executables are stored in a directory hierarchy designated by the `SYSTEM` and `SYSTEM_CPU` environment variables. Symbolic links must be created in the appropriate locations within the *wind*, *wind/pvm*, and *tools* subdirectories. For the case described, the easiest mechanism for sharing the executables is to create a symbolic link between the `XEON` and `OPTERON` directories.

```
ln -s XEON OPTERON
```

The resulting directory structure will be:

```
wind_install/
   wind/
      LINUX64-GLIBC2.5/
         XEON/
         OPTERON -> XEON/
         ...
      bin/
         ...
      pvm/
         lib/
            LINUX64-GLIBC2.5/
               XEON
               OPTERON -> XEON/
               ...
   tools/
      LINUX64-GLIBC2.5/
         XEON/
         OPTERON -> XEON/
         ...
      bin/
         ...
```

where the notation "`OPTERON -> XEON/`" is used to indicate that `OPTERON` is a symbolic link pointing to `XEON/`.

3.3.2 Installation for NFS Access

If multiple users in an organization will be running Wind-US, it is convenient to install the Wind-US and tools executables below a parent directory that can be accessed by all the users via

a network file system (NFS). This way, the location and method of accessing the executables will be the same on all user machines, and they can be centrally updated without any significant effort from individual users. It is suggested that one individual from an organization download the needed application distributions and serve as a single point of contact for Wind-US.

When the scripts and executables are to be accessed via NFS, if the path name used by the user to access the NFS-mounted file system is different from the actual directory name on the "host" system, a slight variation of the above installation procedure is necessary. That's because the user's environment variables `CFDROOT` and `TOOLSROOT` must define the NFS-mounted location of the *wind* and *tools* subdirectories. Similarly, the lines users add to their *.login* or *.profile* files should specify path names used to access the NFS-mounted file system.

NFS file systems are typically accessed using *automount*. As an example, assume that Wind-US and the tools have been installed on a host workstation named *wind_machine*, and that the parent installation directory is */usr/local/wind_nfs*. On *wind_machine*, that directory is exported read-only to the "home" system, normally an individual's workstation, of each Wind-US user. The user can then access that directory via automount by adding "*/net/wind_machine*" to the beginning of the path name. E.g.,

```
cd /net/wind_machine/usr/local/wind_nfs
```

The *cfd.login*, etc., files on *wind_machine* should be copied to new files named *cfd.nfs.login*, etc.[1] The definitions of `CFDROOT` and `TOOLSROOT` in *cfd.nfs.login*, etc., should be modified to point to the automount'ed location of the *wind* and *tools* subdirectories. I.e., in *cfd.nfs.login* `CFDROOT` is defined as

```
setenv CFDROOT /net/wind_machine/usr/local/wind_nfs/wind
```

and in *tools.nfs.login* `TOOLSROOT` is defined as

```
setenv TOOLSROOT /net/wind_machine/usr/local/wind_nfs/tools
```

Finally, using the above example, *csh* and *tcsh* users accessing Wind-US via automount would modify their *.login* file to add the lines

```
source /net/wind_machine/usr/local/wind_nfs/wind/bin/cfd.nfs.login
source /net/wind_machine/usr/local/wind_nfs/tools/bin/tools.nfs.login
```

and *sh* and *ksh* users would modify their *.profile* file to add the lines

```
. /net/wind_machine/usr/local/wind_nfs/wind/bin/cfd.nfs.profile
. /net/wind_machine/usr/local/wind_nfs/tools/bin/tools.nfs.profile
```

Users accessing via NFS would not need to set the `WIND_DEV` environment variable, since they would not be compiling the source code.

3.3.3 Installation for Parallel Execution on Distributed Systems

It should be noted that Wind-US does not need to be installed on each individual machine in order to run in parallel mode using a network of distributed systems. The executables for all the different types of systems and CPUs being used are installed only on the master system. The Wind-US run scripts automatically take care of copying the appropriate Wind-US and PVM executables and files to the systems being used, starting and stopping the message passing software, and cleaning up at the end of the run. See the "Parallel Processing" section of the *Wind-US User's Guide* for details.

[1] This is done so that users logging directly onto *wind_machine* can still access Wind-US as described in the previous sections.

When using MPI for parallel communication, the MPI libraries should be installed on each worker machine.

4 Installing and Running the Build Distributions on the NAS

Pre-compiled executables for Wind-US and/or the tools are not available for the NASA Advanced Supercomputing (NAS) systems: columbia and pleiades. User's must install the build distribution into their local directory. This section describes the procedure for doing this.

4.1 Security Issues on the NAS

User's are reminded that they are responsible for protecting the dissemination of Wind-US, particularly on a shared computer resource like the NAS. It is therefore recommended that users restrict the access permissions on their NAS home and nobackup directories to prevent access from other users. If this is not already the default behavior, one can use the following commands.

- To restrict access to an existing file or directory use the command:
  `chmod go-rwx` *filename*
- To restrict access to all future files and directories created during the current session use the command:
  `umask 077`
- Depending on which linux shell is being used, the `umask` command above can be placed in the *$HOME/.login* or *$HOME/.profile* login scripts to apply to future sessions.

4.2 Storage Issues on the NAS

Storage space on NAS is split between a rather limited *$HOME* directory and a larger */nobackup/$USER* directory. Most users install Wind-US into their home directory and submit jobs from their nobackup directory. Offline tape storage is available by transferring files to the machine called lou. Please see the NAS website (http://www.nas.nasa.gov/) for additional details.

4.3 Computational Resources on the NAS

The NAS computing cluster is comprised of various computing nodes, each of which contains a different number and type of core processors. The user must select which nodes to use by specifying the model name within the PBS script.

Because each processor type has a different computational efficiency, NAS charges for their use via a Standard Billing Unit (SBU). In this scheme, use of faster computing nodes incurs a larger SBU cost. Each submitted job is is given exclusive access to the requested nodes. The user is charged for using each node, even if the job does not utilize all of the available processors. To make the most of their allotted time on NAS, users should try to fully utlize each computing node. The table below summarizes the available computing resources.

Note: The amount of memory available to a PBS job is less than the total physical memory because the system kernel can use up to 4 GB of memory in each node.

For more information, visit:

- http://www.nas.nasa.gov/hecc/support/kb/Resources-Request-Examples_188.html
- http://www.nas.nasa.gov/hecc/support/kb/Pleiades-Configuration-Details_77.html

4.4 Building Wind-US on the NAS

1. Download the Wind-US Build Distribution.

Table 1: NAS Computing Resources (Jan 2024)

Processor Type	Model Name	SBU/node	CPUs/node	RAM(GB)/node
Sandy Bridge	san	0.47	16	32
Ivy Bridge	ivy	0.66	20	64
Haswell	has	0.80	24	128
Broadwell	bro	1.00	28	128
Skylake	sky_ele	1.59	32	192
Cascade	cas_ait	1.64	40	192
Rome	rom_ait	4.06	128	512

This distribution contains all of the necessary run scripts and source files needed to compile and run Wind-US. Registered users are provided the source bundle and/or instructions for downloading it themselves. You should have a file for the form:

```
dist.windus.4.111.tgz
```

2. Transfer the build bundle to NAS (columbia or pleiades).

   Put the build bundle in the (new) directory *$HOME/WINDUS* where the source will be installed. You should now have:

   ```
   $HOME/WINDUS/dist.windus.4.111.tgz
   ```

3. Unpack the build bundle on NAS.

   ```
   cd $HOME/WINDUS
   tar xzf dist.windus.4.111.tgz
   ```

   This should extract everything to the new subdirectory:

   ```
   $HOME/WINDUS/wind-dev
   ```

4. Update the NAS login scripts.

   Users of *csh* and *tcsh* shells must edit the *$HOME/.cshrc* or *$HOME/.tcshrc* file respectively and make sure the following lines appear:

   ```
   module load comp-intel/2020.4.304
   module load mpi-hpe/mpt.2.26
   setenv MPI_NUM_MEMORY_REGIONS 0
   setenv WIND_DEV  "$HOME/WINDUS/wind-dev"
   setenv CFDROOT   "$HOME/WINDUS/wind-dev"
   source            $HOME/WINDUS/wind-dev/bin/cfd.login
   ```

   Similarly, *bash*, *ksh*, and *sh* users must edit their *$HOME/.profile* file and add make sure the following lines appear:

   ```
   module load comp-intel/2020.4.304
   module load mpi-hpe/mpt.2.26
   MPI_NUM_MEMORY_REGIONS = 0                    ; export MPI_NUM_MEMORY_REGIONS
   WIND_DEV  = "$HOME/WINDUS/wind-dev"           ; export WIND_DEV
   CFDROOT   = "$HOME/WINDUS/wind-dev"           ; export CFDROOT
   .            $HOME/WINDUS/wind-dev/bin/cfd.profile
   ```

This sets up the Intel Fortran/C compilers, loads SGI's MPI message passing toolkit, and causes the contents of the *cfd.login* (or *cfd.profile*) file to be executed automatically for each new shell instance. The environment variables `CFDROOT`, `WIND_DEV`, `SYSTEM`, and `SYSTEM_CPU`, that are required for installing and running Wind-US will also be set, and the `PATH` will be modified to include the location of the Wind-US executable directories. Setting `MPI_NUM_MEMORY_REGIONS` to 0 disables the MPI data buffer in order to avoid cache trashing.

Note that the application directory (`CFDROOT`) is set to the same location as the build/development directory (`WIND_DEV`). One could instead use a separate application directory as described in the installation instructions for a general system. However, most NAS users do not need that additional flexibility and want to get the code working as quickly as possible.

5. Test the NAS login scripts.

   At this point, log out and log back in. Check to see if the environment variables have been set to appropriate values, by doing:

   ```
   printenv WIND_DEV
   printenv CFDROOT
   printenv SYSTEM
   printenv SYSTEM_CPU
   ifort --version
   icc   --version
   which mpiexec
   ```

   They should have values similar to:

   ```
   $HOME/WINDUS/wind-dev
   $HOME/WINDUS/wind-dev
   LINUX64-GLIBC2.28
   XEON
   ifort (IFORT) 19.1.3.304 20200925
   icc (ICC) 19.1.3.304 20200925
   /nasa/hpe/mpt/2.26_rhel85/bin/mpiexec
   ```

   Note that `$HOME` will likely be expanded to your full home directory path. If the variables are not set, or not set correctly, go back to step 4 and try again.

6. Move into the Wind-US build directory.

   ```
   cd $WIND_DEV
   ```

   This should put you in the *$HOME/WINDUS/wind-dev* directory.

7. Configure the makefiles.

   If you plan to build both Wind-US and the tools, then follow the instructions for unpacking the tools distribution before making any changes to the makefiles. The reason for this is that the tools distribution also contains a copy of the makefiles and will overwrite any changes you might make here.

   The default values in the configuration files should be sufficient to produce an executable. To be safe, review the contents of the following files or parts of files, where *SYSTEM* and *SYSTEM_CPU* correspond to the `SYSTEM` and `SYSTEM_CPU` environment variables, paying special attention to the items noted below.

   When modifying files it is always a good idea to save a copy of the original (i.e., as **.bak*), and for new files to create an empty **.bak* file. This makes it easy to identify which files you

have changed, and you can use a visual difference program to compare those changes against the original.

- *Makefile.configure*
  - Set `USE_MPI=YES`.
  - Set `USE_PVM=YES`.
  - Set `BUILD_PVM=YES`.
  - The value of `PRECISION` may be changed to make all Fortran variables single or double precision. The default is a mix, with single precision for the mean flow solution and boundary conditions, and double precision for chemistry data, right-hand-side data, grid coordinates, and grid metrics.
- *source/Makefile.user*
  - Defines which of the Wind-US modules should be compiled. Only in rare circumstances does this file need to be modified.
- *source/makefiles/Makefile.include.SYSTEM.SYSTEM_CPU.opt*
  - This file contains the compiler optimizations for the specific system being used.
  - If this makefile does not exist, you will need to create it. Usually the easiest way to do so is to copy and modify one of the existing files. For example:

    ```
    cp -p source/makefiles/Makefile.include.LINUX64-GLIBC2.4.XEON.opt
          source/makefiles/Makefile.include.LINUX64-GLIBC2.28.XEON.opt
    ```

  - Check the definition of `CPP`, which is the full path name for the C pre-processor *cpp*, to make sure it is correct. You can locate *cpp* using:

    ```
    whereis -b cpp
    ```

  - You may need to change the definitions related to the Fortran and C compilers being used (i.e., the variables `ABI`, `FC`, `F90`, `LD`, etc.). There is coding in the Linux makefiles for various Fortran and C compilers. These lines should be commented/uncommented as needed, but additional changes may also be necessary or desired. At the time of this writing, the following Intel 19.1.3.304 compiler settings were used:

    ```
    ABI=        -Zp8 -pc80 -fp-model strict -fno-alias -heap-arrays \
                -fpic -traceback

    FC=         ifort
    FCOMP=      $(FC) $(ABI) -pad -ip -DINTEL
    FFHOP=      -O3 -axCORE-AVX512,CORE-AVX2 -xAVX
    FFOPT=      -O3 -axCORE-AVX512,CORE-AVX2 -xAVX
    FFLOW=      -O1
    FFNOP=      -O0

    F90=        ifort $(ABI) -pad -ip -DINTEL
    F90FHOP=    -O3 -axCORE-AVX512,CORE-AVX2 -xAVX
    F90FOPT=    -O3 -axCORE-AVX512,CORE-AVX2 -xAVX
    F90FLOW=    -O1
    F90FNOP=    -O0
    ```

```
CC=         icc
CCOMP=      $(CC) $(ABI)
ANSI=       -ansi
POSIX=      -D_POSIX_SOURCE
CFOPT=      -O2 -axCORE-AVX512,CORE-AVX2 -xAVX

CCP=        icpc $(ABI)
CPFOPT=     -O2 -axCORE-AVX512,CORE-AVX2 -xAVX

LD=         ifort $(ABI) -O3 -ip -pad -axCORE-AVX512,CORE-AVX2 -xAVX
```

The compiler flag `-ip` applies interprocedural optimizations for single-file compilation. A similar `-ipo` flag is available for multi-file optimizations, but is not used because it takes significantly longer to compile. The `-pad` flag permits the compiler to adjust the location of variables and arrays in memory to improve performance. The compiler flags `-xAVX` and `-axCORE-AVX512,CORE-AVX2` are processor-specific optimizations that set the baseline code path and alternate optimized code paths respectively. For more information, visit:

  * http://www.nas.nasa.gov/hecc/support/kb/Recommended-Compiler-Options_99.html

- If your preferred compiler doesn't support the Fortran long integer data type, add "`-DNOFLONG`" (without the quotes) to the definition of `WINDDEFS`. Errors in the compilation of *mem_management_module.f90* about an ambiguous definition for the generic interfaces `alloc` and `dealloc`, involving the specific interfaces for `allockindlong_1` and `allockindint_1`, etc., are an indication that "`-DNOFLONG`" is needed.

- If necessary, specify the location of the MPI libraries.

```
MPILIBS=        -L/nasa/hpe/mpt/2.26_rhel85/lib -lmpi
```

The SGI MPI module makes available an `mpif90` command that automatically passes the library information to the Intel compiler through the `LD_LIBRARY_PATH`, `LIBRARY_PATH`, and `F_PATH` environment variables. However, all code compiled with this command will include the MPI libraries, even if they are not needed. This has been found to cause problems with some of the tools to be compiled below. Using `ifort` as the compiler directive and setting the `MPILIBS` makefile variable ensures that the MPI libraries will only be included in the Wind-US executable.

8. Compile the source code.

   - Create a PBS script (*$WIND_DEV/make.wind.pbs*) to compile Wind-US. For *csh* and *tcsh* use the following:

```
#PBS -lselect=1:ncpus=4:model=san,walltime=2:00:00
cd $PBS_O_WORKDIR
echo "CFDROOT    = $CFDROOT"      |& tee    make.wind.log
echo "WIND_DEV   = $WIND_DEV"     |& tee -a make.wind.log
echo "SYSTEM     = $SYSTEM"       |& tee -a make.wind.log
echo "SYSTEM_CPU = $SYSTEM_CPU"   |& tee -a make.wind.log
cd $WIND_DEV
make -j 10 opt |& tee -a make.wind.log
```

   For *sh* and *ksh* use the above, but replace "`|&`" with "`2&>1 |`".

- Submit the job to the developer queue to compile the code.

```
cd $WIND_DEV
qsub -q devel make.wind.pbs
```

  This may take roughly 30 minutes once your job begins running. You can use the `qstat` command to check on the status.

- If Wind-US compiled successfully you should now have:

```
$WIND_DEV/$SYSTEM/$SYSTEM_CPU/bin/Wind-US4.exe
```

  If the executable was not produced, then examine the log file (*$WIND_DEV/make.wind.log*) for compilation errors.

9. Install the executables and scripts.
   If you have a previous installation of Wind-US that you wish to update (i.e., `CFDROOT` points to some directory other than the `WIND_DEV` you are building from), you will need to install the Wind-US and PVM executables. To do that, issue the commands

```
make install
make install_scripts
make copy_pvm
```

   This will:

   - Copy the Wind-US executable to: `$CFDROOT/$SYSTEM/$SYSTEM_CPU/bin`
   - Copy the Wind-US run scripts to: `$CFDROOT/bin`
   - Copy the PVM executables and libraries to: `$CFDROOT/pvm/lib/$SYSTEM/$SYSTEM_CPU`
   - Copy the PVM include files to: `$CFDROOT/pvm/include`
   - Copy the PVM scripts to: `$CFDROOT/pvm/lib`

10. If this is a new installation, it would probably be best to log out and log back in before running Wind-US. This executes the shell start-up scripts, modifying the `PATH` environment variable to include the newly-created location for the Wind-US executable.

11. Run the `wind` script.

```
wind
```

    The new executable should appear in the list of available versions.

```
                Select the desired version
0: Exit wind
1: Wind-US 4.0
```

    See Section 4.6 for running Wind-US on the NAS.

4.5 Building the Tools on the NAS

The tools distribution contains source code for all of the Wind-US pre- and post-processing utilities. It is designed to be a completely self-contained package, so one could have separate directory trees for the Wind-US build distribution and the tools build distribution. However, this would lead to a some duplication of shared routines. To reduce this redundancy, the tools distribution is packaged such that it can overlay the Wind-US distribution. The following instructions assume that all of the tools are being built in the same tree as Wind-US.

Note that the build procedure for the tools is somewhat more complicated than the Wind-US build process. Please report problems via the support web page at: https://www.grc.nasa.gov/www/winddocs/support/index.html

1. Download the Tools Build Distribution.
   This distribution contains all of the necessary run scripts and source files needed to compile and run the Wind-US utilities. Registered users are provided the source bundle and/or instructions for downloading it themselves.

   ```
   dist.alltools.tgz
   ```

   Some of the tools require lua and cgnslib header files and/or libraries.

   - Download lua (4.0.1) from `http://www.lua.org/ftp/lua-4.0.1.tar.gz` and save as:

     ```
     lua.4.0.1.tar.gz
     ```

     Due to changes in the API, newer versions of lua will not work with the Wind-US tools!

   - Download cgnslib (2.5.5) from `https://cgns.github.io/download.html` and save as:

     ```
     cgnslib.2.5.5.tar.gz
     ```

     Newer versions of cgnslib may also work.

2. Transfer the source code to NAS (columbia or pleiades).

   - Put the **gz* files in the (existing) directory *$HOME/WINDUS* where the source will be installed. You should now have:

     ```
     $HOME/WINDUS/alldist.tools.tgz
     $HOME/WINDUS/lua.4.0.1.tar.gz
     $HOME/WINDUS/cgnslib.2.5.5.tar.gz
     ```

3. Unpack the source code on NAS.

   Note that this will overwrite any changes you made to the Wind-US makefiles!

   ```
   cd $HOME/WINDUS
   tar xzf dist.alltools.tgz
   gunzip -c lua.4.0.1.tar.gz | tar xvf -
   gunzip -c cgnslib.2.5.5.tar.gz | tar xvf -
   ```

   This should extract everything to the following subdirectories:

   ```
   $HOME/WINDUS/wind-dev/tools-dev/*
   $HOME/WINDUS/lua-4.0.1
   $HOME/WINDUS/cgns_2.5
   ```

4. Update the NAS login scripts.

   Users of *csh* and *tcsh* shells must edit the *$HOME/.cshrc* or *$HOME/.tcshrc* file respectively and make sure the following lines appear:

   ```
   module load comp-intel/2020.4.304
   module load mpi-hpe/mpt.2.26
   setenv MPI_NUM_MEMORY_REGIONS 0
   setenv WIND_DEV  "$HOME/WINDUS/wind-dev"
   setenv CFDROOT   "$HOME/WINDUS/wind-dev"
   setenv TOOLSROOT "$HOME/WINDUS/wind-dev/tools-dev"
   ```

```
source            $HOME/WINDUS/wind-dev/bin/cfd.login
source            $HOME/WINDUS/wind-dev/tools-dev/bin/tools.login
```

Similarly, *bash*, *ksh*, and *sh* users must edit their *$HOME/.profile* file and make sure the following lines appear:

```
module load comp-intel/2020.4.304
module load mpi-hpe/mpt.2.26
MPI_NUM_MEMORY_REGIONS = 0 ; export MPI_NUM_MEMORY_REGIONS
CFDROOT   = "$HOME/WINDUS/wind-dev"           ; export CFDROOT
WIND_DEV  = "$HOME/WINDUS/wind-dev"           ; export WIND_DEV
TOOLSROOT = "$HOME/WINDUS/wind-dev/tools-dev" ; export TOOLSROOT
.            $HOME/WINDUS/wind-dev/bin/cfd.profile
.            $HOME/WINDUS/wind-dev/tools-dev/bin/tools.profile
```

This causes the contents of the *tools.login* (or *tools.profile*) file to be executed automatically at login time to set up the environment variable `TOOLSROOT` that is required for installing and running the Wind-US tools, and to modify `PATH` to include the location of the proper executable.

5. Test the NAS login scripts.

   At this point, log out and log back in. Check to see if the environment variables have been set to appropriate values, by doing:

   ```
   printenv WIND_DEV
   printenv CFDROOT
   printenv TOOLSROOT
   printenv SYSTEM
   printenv SYSTEM_CPU
   ifort --version
   icc   --version
   ```

   They should have values similar to:

   ```
   $HOME/WINDUS/wind-dev
   $HOME/WINDUS/wind-dev
   $HOME/WINDUS/wind-dev/tools-dev
   LINUX64-GLIBC2.28
   XEON
   ifort (IFORT) 19.1.3.304 20200925
   icc (ICC) 19.1.3.304 20200925
   ```

   (Note that `$HOME` may be expanded to your full home directory path.)

   If the variables are not set, or not set correctly, go back to step 4 and try again.

6. Compile Lua.

   ```
   cd $HOME/WINDUS/lua-4.0.1
   make
   ```

   This should create the following files:

   ```
   $HOME/WINDUS/lua-4.0.1/bin/lua
   $HOME/WINDUS/lua-4.0.1/bin/luac
   $HOME/WINDUS/lua-4.0.1/lib/liblua.a
   $HOME/WINDUS/lua-4.0.1/lib/liblualib.a
   ```

7. Compile the CGNS library.

```
cd $HOME/WINDUS/cgnslib_2.5
./configure --prefix=$HOME/WINDUS/cgnslib_2.5 --with-system=LINUX64 \
            --enable-64bit
make SYSTEM=LINUX64
mkdir include
mkdir lib
make install
```

   This should create the following files:

```
$HOME/WINDUS/cgnslib_2.5/include/cgnslib.h
$HOME/WINDUS/cgnslib_2.5/include/cgnslib_f.h
$HOME/WINDUS/cgnslib_2.5/include/cgnswin_f.h
$HOME/WINDUS/cgnslib_2.5/lib/libcgns.a
```

8. Now that the environment variables are set properly, move into the Wind-US build directory.

```
cd $WIND_DEV
```

   This should put you in the *$HOME/WINDUS/wind-dev* directory.

9. Configure the makefiles.

   Review the contents of the following files or parts of files, where *SYSTEM* and *SYSTEM_CPU* correspond to the `SYSTEM` and `SYSTEM_CPU` environment variables, paying special attention to the items noted below.

   - *Makefile.configure*
     - Use the same settings described above for building Wind-US on NAS.
   - *source/makefiles/Makefile.include.SYSTEM.SYSTEM_CPU.opt*
     - Make the modifications listed above in the instructions for building Wind-US on NAS.
     - Comment out the explicit static and shared library settings:

```
#TOOLS_STATLIB= -static
#TOOLS_SHARLIB= -shared
```

       The default behavior of the Intel compiler is to use shared libraries.

     - Provide the proper locations of Tcl, Lua, and CGNS:

```
TOOLS_TCLLIBS=  -L/usr/lib64 -ltcl8.5
TOOLS_LUALIBS=  -L$(HOME)/WINDUS/lua-4.0.1/lib -llua -llualib
TOOLS_CGNSLIBS= -L$(HOME)/WINDUS/cgnslib_2.5/lib -lcgns
TCL_INCLUDE=    $(INCCMD)/usr/include
LUA_INCLUDE=    $(INCCMD)$(HOME)/WINDUS/lua-4.0.1/include
CGNS_INCLUDE=   $(INCCMD)$(HOME)/WINDUS/cgnslib_2.5/include
```

     - Provide the proper locations of the Intel libraries:

```
TOOLS_OSLIBS=   -L/nasa/intel/Compiler/2020.4.304/lib/intel64 \
                -lifcore -lirc -limf -lpthread
TOOLS_CPPLIBS=  -L/nasa/intel/Compiler/2020.4.304/lib/intel64 \
                -lifcore -lirc -limf
```

```
MADCAP_OSLIBS=  -L/nasa/intel/Compiler/2020.4.304/lib/intel64 \
                -lifcore -lcxa -lunwind -lpthread -lstdc++
```

Depending on which version of the Intel compiler you choose, the library files may be in a different subdirectory below */nasa/intel*. These library paths might already be present in the enviroment variables.

- *source/makefiles/pvm_conf/SYSTEM.SYSTEM_CPU.def.opt*
  - Do not bother configuring or creating this file unless you encounter an error during the compilation process. This file is used to override the default settings given in a similarly named *pvm/conf/SYSTEM.def* file.
- *tools-dev/Makefile*
  - This file defines the make dependencies and should not need to be changed.
- *tools-dev/libmdgl/SYSTEM.mkf*
  - This file is used to compile the graphics library needed by GMAN, CFPOST, and MADCAP.
  - If this makefile does not exist, you will need to create it. Usually the easiest way to do so is to copy and modify one of the existing files.
  - Use the settings for the Intel compiler, if they are not already the default.

    ```
    ABI=    -mcmodel=medium -pad -pc80 -fp-model strict -fno-alias
    CC=     icc
    CFLOPT=
    ```

    Note that `CFLOPT` is defined to be empty.

10. Compile the tools source.

    On NAS, the front end node has the openmotif-devel-* package installed, which makes available a number of header files needed to compile the Wind-US tools. The worker nodes only have the library files installed. This means that the tools must be compiled on the front end node. So, instead of using a batch script like that to compile Wind-US, simply compile the tools from the command line.

    Make sure you are in the build directory.

    ```
    cd $WIND_DEV
    ```

    Next, *csh* and *tcsh* users should do

    ```
    make all_tools |& tee make_tools.log
    ```

    while *sh* and *ksh* users should do

    ```
    make all_tools 2>&1 | tee make_tools.log
    ```

    Tools can also be compiled individually, by doing

    ```
    make tool_name
    ```

    where *tool_name* is the name of the tool. Note that the names to be used for GMAN, CFPOST, and MADCAP are *gmanpre*, *cfpost_prod*, and *madcapprod*, respectively.

    After compilation is complete, the following programs should appear in directory *$WIND_DEV/$SYSTEM/$SYSTEM_CPU/bin*

```
USintrpltQ.exe  cfpost_prod.exe   chmgr.exe      mpigetnzone  thplt.exe
adfedit         cfreorder.exe     decompose.exe  newtmp       timplt.exe
cfappend.exe    cfreset_iter.exe  delvars        npair        tmptrn.exe
cfaverage.exe   cfrevert.exe      fpro           parcnl       usplit-hybrid.exe
cfbeta.exe      cfsequence.exe    gman_pre.exe   readcf       windpar.exe
cfcnvt          cfspart           gpro.exe       resplt.exe   wnparc
cfcombine.exe   cfsplit.exe       gridvel.exe    rnparc       wplt3d
cflistnum       cfsubset.exe      icees          rplt3d       writcf
cfnav.exe       cfunsequence.exe  jormak.exe     scan
cfpart.exe      cfview.exe        lstvars        terp
```

Depending on the size of the array parameters requested, the `thplt.exe` executable might not get created with the default memory model. If it was not created, then edit the file *$WIND_DEV/source/makefiles/Makefile.include.$SYSTEM.$SYSTEM_CPU.opt* to use the following ABI settings:

```
ABI=       -Zp8 -pc80 -fp-model strict -fno-alias -heap-arrays \
           -mcmodel medium -traceback
```

and recompile just that tool. From the build directory, remove the old object file.

```
cd $WIND_DEV
rm -f OBJECTS/$SYSTEM/$SYSTEM_CPU/thplt.o
```

Next, *csh* and *tcsh* users should do

```
make thplt |& tee make_thplt.log
```

while *sh* and *ksh* users should do

```
make thplt 2>&1 | tee make_thplt.log
```

Check *$WIND_DEV/$SYSTEM/$SYSTEM_CPU/bin* to confirm that `thplt.exe` was created.

11. In order for the tool scripts to locate the executables, they must be installed in the proper location. To install the executables, do:

    ```
    make install_tools
    ```

    This copies the tools executables to *$TOOLSROOT/$SYSTEM/$SYSTEM_CPU/bin*.

12. If this is a new installation, it would probably be best to log out and log back in again before running any of the tools. This executes the shell start-up scripts, modifying the `PATH` environment variable to include the newly-created location for the tools executables.

4.6 Running Wind-US on the NAS

1. Make a directory containing your Wind-US input files:

   ```
   run.dat
   run.cgd
   run.mpc
   run.lis  (if continuing from a previous solution)
   run.cfl  (if continuing from a previous solution)
   ```

   The *run.mpc* file should have the following form:

   ```
   / Wind-US parallel processing file for NAS.
   / Currently set to use 2 nodes with 20 processors each
   ```

```
/                 and 1 node  with  6 processors
/                 for a total of   46 cores.
/
host localhost nproc 20
host localhost nproc 20
host localhost nproc 6
```

Each different type of NAS compute node has a different number of processors. For example, the Ivy Bridge nodes have 20 processor cores. Therefore, each host entry above has at most `nproc 20`. The user will need to experiment to determine whether the best performance is obtained when all of the processor cores on a given host are used (20+20+6=46) or when the hosts are most closely balanced (16+16+14=46). The difference between internode and intranode communication might be mesh dependent.

Users might also want to include a `checkpoint` command in the above file so that the worker solutions are sent to the master process at regular intervals. Please see the Wind-US User's Manual for more details on the format and features of the parallel processing file.

2. Start the Wind-US script with one of the following commands:

```
wind -runinplace -cl -usessh -mpmode MPI -mpiver SGI
wind -runinplace -cl -usessh -mpmode PVM
```

- To specify a non-default NAS charge number (ie, a0101), add the following to the Wind-US command line.

  ```
  -grpcharge a0101
  ```

  To obtain a list of the groups you have access to, type:

  ```
  groups
  ```

- Follow the prompts for your input, output, mesh, and flow files or specify them on the command line using the proper syntax.
- When asked, indicate that you want to run in multi-processor mode.
- If prompted, enter the number of zones in the cgd file.
- When prompted for the type of queue, choose QSUB_PBS_QUE.
- Enter the queue name: i.e., `long` for the long queue.
- Enter the solver run wall clock time.
- Enter the termination processing time.
- Enter the number of nodes to run on. This should match the number of `host` entries in your mpc file.
- Enter the number of processors per node to use.
- Enter the number of MPI processes per node to use. This is typically the same as the number of processors per node. For the default setting of `ASSIGNMENT MODE DEDICATED` in the *.mpc* file, each zone should have its own MPI process. One additional MPI process is required for the master process. You must remember to account for this extra process when answering the prompts or your run will fail to initialize MPI.
- Enter any optional attributes. For example, to specify that the job should run on Ivy Bridge nodes use the following:

```
model=ivy
```

The model names for other processor types are listed in Table 1.

- When prompted to "Press CR to submit job, or another key (except space) and CR to abort," do the latter. This will create a file called *run.job.pl*.

The maximum solver execution time is determined by subtracting the termination processing time from the solver run wall clock time. When the Wind-US run job is submitted, it will create a *preNDSTOP* file. When the maximum solver execution time has expired, this file will be renamed *NDSTOP*, forcing Wind-US to begin a graceful shutdown.

Users should make sure that the termination processing time is sufficient to allow Wind-US to complete the termination process. At the end of the **.lis*, there is a summary indicating the time spent during execuation and termination.

Users should also make sure that they request adequate time from the queue in which they submit their jobs. This is detailed in the next step. Otherwise, the queue will terminate Wind-US, resulting in a less than graceful shutdown.

3. Edit *run.job.pl*. If you see a line like the following near the top of the file:

```
#PBS -l nodes=1234:ppn=2
```

replace it with

```
#PBS -l select=2:ncpus=20:model=ivy+1:ncpus=6:model=ivy
#PBS -l walltime=40:00:00
#PBS -m e
```

This will select 2 nodes with 20 cpus and 1 node with 6 cpus, which matches the request in the *run.mpc* file. The walltime can be adjusted as desired (hh:mm:ss) or as an integer number of seconds, and the last line will send you an email when your job is completed.

Make sure to request at least as much walltime as was specified in the Wind-US prompts above, because the queuing system does not terminate jobs as cleanly as Wind-US does.

4. If you plan on resubmitting the same job again later (i.e., you want to run 10000 cycles now and 10000 cycles later) you can save a copy of the run script.

```
cp -p run.job.pl run.job.pl.bak
```

To resubmit later, you can skip the above steps and simply reuse the run script.

```
cp -p run.job.pl.bak run.job.pl
```

Note that if you increase the number of cycles in your **.dat* file, you may need to adjust the run time specified in the job file. In this case it might be best to answer the Wind-US prompts again to create a new `run.job.pl` file.

5. Submit the job to the `long` queue with the command:

```
qsub -q long run.job.pl
```

Some other useful commands are:

Command	Action
`node_stats.sh`	List the number and type of available cpu nodes.
`qstat -q`	List all queue names and run limits.
`qstat -a long`	List all jobs running in the `long` queue.
`qstat -u USER`	List all jobs running for username `USER`.
`qdel JOBNAME`	Delete job with name `JOBNAME`. Useful if Wind-US has not yet started.

6. In order to improve I/O performance for large jobs, Wind-US 3.0 (and later) uses a newer ADF library than its predecessors. Grid and solution files used with Wind-US will automatically be upgraded to the new format, and the tools compiled in the above steps will also work with the new file structure. However, if you transfer the grid or solution file(s) back to your local workstation your existing tools may not be able to read them. If you experience this problem, you should upgrade the tools at your local site.

5 Porting Wind-US to a New UNIX Platform[2]

5.1 Porting Guidelines

The source files for Wind-US (and the non-system libraries it depends on) are provided in the build distribution. See Section 2 for instructions on how to obtain a copy. Put the gzip'ed tar file containing the build distribution into an appropriate directory. The same one used for the application and tools distributions would be a good choice. In that directory, unpack the file by doing:

```
gunzip -c filename | tar xvf -
```

where *filename* is the file name, including the *.tar.gz* extension. Once you have installed the source on your system, change directory to the *wind-dev* directory and set the `WIND_DEV` environment variable to point to that directory. For example, if the previous step was done in */usr/local/wind*, *csh* and *tcsh* users would do

```
cd wind-dev
setenv WIND_DEV /usr/local/wind/wind-dev
```

Now you are ready to begin the task of actually porting Wind-US. There are three files you need to either examine or, if they don't already exist, create:

- *$WIND_DEV/Makefile.Configure*

  This file contains generic information such as the name of the *make* utility, where to find the source for the various components of Wind-US, and whether or not to build PVM. In addition, this file defines the machine and CPU type for which Wind-US will be built. By default, these are set from the `SYSTEM` and `SYSTEM_CPU` environment variables. If this file doesn't exist, download the build distribution again — something went very wrong.

- *$WIND_DEV/source/makefiles/Makefile.include.$SYSTEM.$SYSTEM_CPU.opt*

  This file contains system-specific information, such as compiler names, optimization switches, machine-specific compilations, the name of the *awk* command, the location of the C pre-processor *cpp*, and extra libraries which must be linked in. Pay particular attention to the `MCHNSRCS` lines in the "Common File directory special rules" section. If you are compiling for a completely new machine, you may have to create one or more of these files (which are found in *$WIND_DEV/libcfd/machine.lib*). You may, however, be able to use files created for other machines.

  Note: There are other possible extensions than *.opt* for this file. If you wish to compile for use with a debugger, create a *Makefile.include...* with a *.dbx* extension. See the SGI files for examples. Another possible extension is *.pure*, which defines the compilation configuration for use with the *Purify* debugging software package.

- *$WIND_DEV/source/makefiles/pvm_conf/$SYSTEM.$SYSTEM_CPU.def.opt*

  This is the PVM configuration file. If your system is not currently supported, check in *$WIND_DEV/pvm/conf* to see if a configuration file for your system already exists. Note that the definition of `SYSTEM` may be different for PVM than for Wind-US. If you cannot find a pre-existing configuration file for your system, you will have to create one. Choose a file for a system similar to yours and use that as a starting point.

  Note: As before, there are other extensions besides *.opt*. See the SGI files for examples.

[2]The material in this section was originally written by Chris Nelson of Sverdrup Technology, Inc. - AEDC Group.

When all three files exist and everything appears in order, then simply type `make opt` (if you want to compile for optimization) and the make system should take care of the rest. If you type `make` by itself (or `make help`), a list of all the compilation options will be printed.

5.2 Frequently (or not) Encountered Problems

1. *The `SYSTEM` and `SYSTEM_CPU` variables are not being set in a way that makes sense for my system. What can I do?*

   It is likely that the *pvmgetarch* and *pvmgetcpu* scripts need to be modified to detect your particular system. These scripts are found in the *$CFDROOT/bin* directory. You will have to determine for yourself how the scripts can correctly distinguish your system from others, but the examples already in them should get you started.

   Once you have modified the files, you will need to copy them to several different places. Copy *$CFDROOT/bin/pvmgetarch* to *$CFDROOT/pvm/pvmgetarch-cfd* and also to *$WIND_DEV/pvm/lib/pvmgetarch-cfd*. Copy *$CFDROOT/bin/pvmgetcpu* to *$CFDROOT/pvm/pvmgetcpu-cfd*.

   Ideally these files should be links, and perhaps some day they will be.

2. *Can I cross-compile Wind-US for a different system using these makefiles?*

   If you have a compiler that is capable of compiling for many different machines/CPUs, then you may want to use a single machine to create executables for all of them. To some extent, this is possible, but it will take some extra work on your part.

   First, modify *$WIND_DEV/Makefile.configure* so that `SYSTEM_SUFFIX` and `SYSTEM_BLD_CPU` are set to the machine you wish to compile for. The extra work comes because the PVM compilation system is not set up for cross-compiling. Therefore, you must set `BUILD_PVM` to `NO` and obtain (by one means or another) PVM libraries and executables for each machine you wish to compile for. For SGI workstations with MIPS processors, the default is to compile PVM for the lowest common denominator CPU, so, for a given operating system, you should be able to use the same PVM files for R10000 (and up) machines.

3. *I only have a single processor machine. Do I really have to mess with all this parallel stuff?*

   No, you don't. To turn off the parallel capabilities of Wind-US, edit *$WIND_DEV/Makefile.configure* and set `BUILD_PVM` to `NO`. Next, edit *$WIND_DEV/source/Makefile.user* and remove `pssubs` from the `LINK_MODULES` line and add it to the `DUMMY_MODULES` line.

4. *The compilation gets all the way to the end, and then fails with complaints about `rcutv1`, `rcutaa`, and `rcimsc` being unresolved symbols. What gives?*

   This problem pops up on Sun workstations (and maybe others) because *awk* is not working as expected. Check to see if *nawk* is available on your system. If it is, edit *$WIND_DEV/source/makefiles/Makefile.include.$SYSTEM.$SYSTEM_CPU.opt* and set the `AWK` variable to `nawk`.

5. *I modified $WIND_DEV/source/makefiles/Makefile.include... (or $WIND_DEV/source/makefiles/pvm_conf/$SYSTEM...def...), but when I re-compile, none of my changes are picked up. What is wrong?*

   The problem is that the files that are actually used for the compilation are not the ones under *$WIND_DEV/source/makefiles*. The actual files are: *$WIND_DEV/Makefile.include.$SYSTEM.$SYSTEM_CPU* and

$WIND_DEV/pvm/conf/$PVMSYS.def, where *$PVMSYS* is set by *$WIND_DEV/pvm/lib/pvmgetarch*. When you make changes, you must either copy the files to their final destination or "select" the makefiles for the type of build you're doing (using, for example `make select_opt` if you want to compile with optimization). When you run one of the "global re-build" compilations (e.g., `make opt`) then the "select" is done automatically. Ideally, the system should automatically check to see if any of the configuration files have been modified, but right now they don't.

6. *I modified $WIND_DEV/Makefile.include.SYSTEM.SYSTEM_CPU (or $WIND_DEV/pvm/conf/$PVMSYS.def), but when I tried to build Wind-US, it didn't seem to find my changes. I checked the files, and my changes were gone. What happened?*

   See the answer to #5, above. The short answer is that your changes were overwritten. You have to modify the files under *$WIND_DEV/makefiles* and "select" them in order to be sure that the changes will "stick".

7. *My make utility complains that there are errors and aborts before anything gets compiled. Why?*

   IBMs seem to be particularly bad about this. The "errors" are usually not errors in the sense that anything is catastrophically wrong, but rather, in the process of house-cleaning, the make system may be trying to remove files that don't exist or is checking to see if a file does exist when it doesn't. The solution is to modify *$WIND_DEV/Makefile.configure* and add a `-i` switch to the `MAKE` and `PVMMAKE` variables. Sometimes, switching to *gmake* will solve the problem, but be aware that the PVM make system has *make* hardwired. It may also be necessary to start the make process with the `-i` switch (e.g. `make -i opt`).

8. *When I try to compile, it gets to a certain point and then just hangs. What is the problem?*

   The system of makefiles used to build Wind-US is pretty complex, and some versions of *make* just can't handle this complexity. The weakest link appears to be in the PVM build. If your *make* utility is not up to the task, try using another one (such as *gmake*). Since the PVM makefiles are hardwired to use *make*, you may have to use an alias rather than changing *Makefile.configure*. For example, on the HP/Convex CSPP system, it might be necessary to alias *make* to *gmake*.

9. *I'm getting undefined symbols at the end of my compilation, but it's more than just the three you mentioned in question #4. What are likely causes of the problem?*

   The most likely explanation is that your system does not load all the libraries you need by default. Run *grep* on some of the symbols that it complains about (ignoring post-pended underscores — i.e. `psexit`, not `psexit_`) and see if the routines are from within Wind-US:

   ```
   cd $WIND_DEV/source
   grep the_unknown_symbol */*
   ```

   If that fails to find anything, check in *libcfd*:

   ```
   cd $WIND_DEV/libcfd
   grep the_unknown_symbol */*
   ```

   If you still can't find it, try *libadf*:

   ```
   cd $WIND_DEV/libadf
   grep the_unknown_symbol *
   ```

   If that fails as well, then hunt through the *$WIND_DEV/pvm* subdirectories in a similar fashion. Once you are satisfied that the symbol is not from anything in the Wind-US

distribution, you will have to poke around the system libraries to identify which ones should be added to the `EXTRALIBS` line in *$WIND_DEV/source/makefiles/Makefile.include.$SYSTEM.$SYSTEM_CPU.opt* (or *.dbx* etc.).

If the symbols are from within Wind-US, you need to look back through the compiler listing to see what messages were output when the library which contains that routine was compiled to see what went wrong.

10. *I'm having trouble compiling the ADF library. What can I do?*

    The ADF core library (*libadf*) has only been ported to a finite number of machines. If your machine is not one of them, you may have to add some lines to *ADF_fbind.h* for your system.

11. *I'm getting some fairly bizarre compiler errors when compiling the Common File library. What is going on?*

    As with the ADF library (see #10), the Common File library has only been ported to a limited number of machines. Check in *$WIND_DEV/libcfd/include/bind_f_and_c.h* to see if definitions for your system are there. If not, you will have to add them.

12. *Okay, I got libadf and libcfd to compile, but now Wind-US itself is complaining. Where should I look?*

    As with *libadf* (see #10) and *libcfd* (see #11), you may need to add lines appropriate for your system to a header file. In this case, it's *$WIND_DEV/source/include/fbind.h*.

13. *I'm having trouble getting PVM to compile and run properly. What should I do?*

    If the problem seems to be with the *make* system itself, then you may be able to successfully compile "manually" by using the following procedure (modify as needed for your particular shell):

    ```
    cd $WIND_DEV/pvm
    setenv PVM_ROOT $cwd
    make clean
    make
    ```

    If that works, then copy the PVM libraries (in *$WIND_DEV/pvm/lib/$PVMSYS*) to the `$LIBDIR` defined in *Makefile.configure*. Also remember to copy the PVM executables to *$CFDROOT/pvm/$SYSTEM/$SYSTEM_CPU*.

    An additional possibility is that you have another version of PVM already installed, and certain environment variables may be set for that version which conflict with the version of PVM shipped with Wind-US. The solution is to make sure that neither of the environment variables `PVM_ROOT` nor `PVM_ARCH` are set prior to compiling or running Wind-US. (The various scripts should set these variables as needed).

    If you still can't compile or run PVM, then you may need to talk to the PVM developers (see http://www.epm.ornl.gov/pvm/) to see about porting it to your system. In the meantime, you can still run Wind-US in single processor mode (see #3).

14. *I've gone through the whole process you describe above (in Section 5.1), but when I type* `make opt`*, I get one or more errors dealing with file permissions, like*

    ```
    cp: cannot create /usr/local/wind/wind-dev/include/ADF.h:
    Permission denied.
    ```

What is the problem?

By default (for security reasons, I believe), the source files only have "read" permission enabled. Thus, when the make system tries to do an operation which results in a "write" (which happens mostly when it's setting up the *include* directory), an error results and the system screeches to a halt. The solution is to make sure that you have write permission for all files and directories under *$WIND_DEV*.

15. *When running the Wind-US code on my brand new SGI R12000 system, I get the following error message*

```
 Program aborting due failure in common I/O library call.
 Subroutine called: CFRWFC
ADF 54: A node-id of 0.0 is not valid.
```

What should I do?

The cause of this problem has been traced to a change in the default floating point exception mode for R12000 systems. R10000 and R4400 SGI systems are not affected.

To run Wind-US on R12000 systems, you can upgrade to IRIX 6.5.4, and make sure that the kernel parameter "`fpcsr_fs_bit`" is equal to zero. After upgrading to IRIX 6.5.4, the value of this parameter may be determined by doing

```
systune fpcsr_fs_bit
```

If the value is non-zero, it should be changed by doing (as root)

```
systune fpcsr_fs_bit 0
```

The change in the value of `fpcsr_fs_bit` occurs dynamically, and does not require rebooting the system.

16. *I finally got everything to compile and link, but when I try to run the code, I get a "Program aborting due failure in common I/O library call" message, and then the code exits. What should I do?*

If you're running on an SGI R12000 system, see the previous question. Otherwise ...

Ironically, the weakest link in the Wind-US chain (as far as porting goes) has nothing to do with the solver algorithm, parallelization, or memory management (in the fluid solver). The weakest link is, in fact, the file I/O system. The problems all seem to center around the ADF core library. Even if you did not see a specific ADF error code (e.g., "ADF 54: A node-id of 0.0 is not valid"), if you can run the same case (with the same files) on another machine, the problem is almost certainly in *libadf*.

If this happens, please notify the code developers so that we can work on finding a fix. Obviously, if we don't have access to a machine of the type you are working on, then our ability to solve the problem is limited, but at least we can note the problem. If you feel energetic, you could also notify the CGNS folks over at https://cgns.github.io/ that you found a problem.

You can, of course, attempt to debug it yourself. If you do find a solution, please send it to us so that we can make the fix available to everyone. If, however, you don't have the time or the inclination for that, then your best bet is to convert your input Common Files (which are version 3.0 by default) to version 2.0. Version 2.0 of the Common File system does not use the ADF core, and, so far, it has always worked when version 3.0 wouldn't. The way to convert

the files is to use the *cfcnvt* utility. Choose option 3, "Compress a Common File", answer the questions, and then enter "2" when asked "Output CF version number (2 or 3 (default))".[3]

17. *When compiling the Common File library, I'm getting messages about an undefined function called "tempnam". What should I do?*

    Edit all *Makefile.include.$SYSTEM.$SYSTEM_CPU.** files (found in *$WIND_DEV/makefiles*). Look for a line that defines "CFDEFS". On this line add "-DNO_TEMPNAM". Next, "select" the appropriate compilation type (e.g., `make select_opt`) and re-compile.

[3] Note that this assumes you have access to a system for which the tools executables, including *cfcnvt*, are available.