POP INSTALLATION GUIDE

Supported System Procedures

Bringing up POP on a supported system

Machines on which POP is supported

POP is written in Fortran 90 and uses two-dimensional domain decomposition (in longitude and latitude) with MPI (Message Passing Interface). This allows POP to be run on a wide variety of parallel computers as well as workstation-class computers. POP is highly portable to any parallel computer that supports Fortran 90 and MPI. The machines on which POP and its 'makefile' system have been tested are listed below. Usually only minor tweaks are required to get POP running on a new architecture, but sometimes more is required.

Hardware	O/S	Fortran	MAKE	Serial	MPI	SHMEM	SMP
SGI Origin 2000	Irix 6.5	MIPSpro F90	gmake 3.77	yes	yes	yes	no
Pentium II	Linux	PGI mpif90	gmake 3.77	yes	yes	no	no
Cray T3E	Unicos	F90	gmake 3.77	yes	yes	no	no
Pentium II	Win NT	Digital Visual 5	built into IDE	yes	no	no	no

Where to get help

CSM-POP users

If you are using POP as a component of the NCAR CSM, technical support is provided by Matthew Hecht (hecht@ncar.ucar.edu, 303-497-xxxx).

LANL-POP users

If you obtained POP directly from LANL and are involved with its development, technical support will be provided through John Davis (jfd@lanl.gov, 505-667-4793) and Phil Jones (pwjones@lanl.gov, 505-667-6386).

Where to get the source code

LANL ACL website

The latest version of the POP source code is always available through http://www.acl.lanl.gov/climate. POP is distributed as a compressed tar file named POP_nnn.tar.Z, where 'nnn' is the version number. An accompanying ASCII text file named POP_nnn_Readme describes the changes made in this version.

NCAR CSM website

It is anticipated that POP will be adopted as the ocean component of the Climate System Model, at which time the version of POP that is officially supported by CSM will become available from the NCAR website http://www.ncar.ucar.edu.

Setup procedure

Extracting source code

It is a simple procedure to get and extract the POP source files. Go to the directory where you want to store the POP tar file. Get the compressed tar file, POP_nnn.tar.zzz from the FTP server. If the file name ends with 'gz', uncompress it with a 'zip' utility (eg., 'gunzip' on Unix or Linux). If it ends with 'Z', use the Unix standard 'uncompress'. You will end up with POP_nnn.tar. As you can verify with "tar tf POP_nnn.tar", the paths within the tarfile are relative, each beginning with "./".
Find a file system with at least ten megabytes free and create a working directory named anything you like, ***poptop*** for example.
Change directory to ***poptop*** and type "tar xf <path-of-**POP_***nnn*.tar>". In a short time you will have a tree something like this:

```
.../poptop/
       input_templates/
       setup_run_dir*
       setup_test_dir*
       mpi/
       serial/
       shmem/
       source/
       tools/
```

Setting up a working directory

A working directory is created by the C shell script *setup_run_dir* . You invoke it at the same directory level by typing

setup_run_dir directory [model]

where

directory is an arbitrary working directory name,
model is the kind of run for which you want to build POP.

(The default model is test; other possibilities are sector and 192x128x32.)

Setting environment variables

What happens in any working directory is determined by a single environment variable, ARCHDIR. Primarily, ARCHDIR sets the configuration for which POP is built. The configuration has two parts, the first for the ARCHitecture (sgi, unicos, linux, etc.), the second for the DIRectory (mpi, serial, shmem) containing code variants.
For example, ARCHDIR=sgi_mpi means that pop will run under Irix in parallel mode using MPI . ARCHDIR=sgi_serial means that pop will run under Irix on a single SGI processor in serial mode.

Parallel/vector/cache architecture options

Total number of processors

Pick an even number. The number depends on the available hardware arrangement.

X-Y assignment of processors

Factor the total number of processors into NPROC_X and NPROC_Y for the X and Y dimensions respectively. Edit the definitions in the working directory copy of ARCHDIR.gnu.

Configuring the make system for your machine

The following files are under every working directory as part of the make system. They are automatically invoked as needed by the gmake command.

fdepends.awk - *awk file for defining dependencies of .f files on other .f files containing modules.*

GNUmakefile - *gmake top-level control file which 'includes' or references others in this list.*

sgi_serial.gnu - *'included' when building a one-processor version of POP on Bluemountain.*

sgi.gnu - *'included' when building an MPI multiprocessor version on Bluemountain.*

sgi_shmem.gnu - *'included' when building a shared memory multiprocessor version on Bluemountain.*

t3d_serial.gnu - *'included' when building a one-processor version on a Cray T3D or T3E.*

utilities.gnu - *'included' to define target "clean".*

compile/GNUmakefile - *referenced by level above. Originally designed to allow building while in the compile subdirectory, but that capability is no longer supported. (It still works, but it may go away in future versions.)*

Building a POP executable

POP comes with a test problem built in. You should run it to see if POP will work correctly on your machine. Even on a supported system, your particular machine configuration may require you to edit existing files.
Let's walk through an example. Suppose you are building on a cluster of Pentium Linux workstations which has MPI already installed. You've untarred the POP distribution under a directory called ***pop***.

Create a working directory for a test problem

You need a clean working directory directly under ***pop***. Pick a name that doesn't already exist, say *trial*. With your default directory as ***pop***, type "*setup_run_dir trial test*". You will see a screenful of informative messages ending with "Copying ../trial/compile/GNUmakefile".

Building POP normally

Make sure your environment variable ARCHDIR is set to "linux". Change to the ***trial*** directory you just created. Type "gmake". Wait. When gmake finishes, your working directory ***trial*** will have an executable file named *pop*. You are ready to run.

Building a debug version

Debugging a parallel code is not always easy and is probably not the best way to solve portability problems with POP. Inserting print statements into the source code and recompiling is usually faster. But if you must, here is how to do it.
To build a debug version, ***pop_db***, type "gmake OPTIMIZE=no". ("gmake" by itself will build the optimized executable ***pop***.) Object files are no longer archived in a separate library, so it is necessary to do "*gmake clean*" when switching between optimized and non-optimized versions.

Running the test problem to verify the installation

type "mpirun –np 4 pop > *somefilename.out*". It should finish in a minute or so. Compare *somefilename.out* with the read-only sample output file "pop_sgi.log" in the same directory. Most systems will have "diff" and "xdiff" available for comparing these text files. A slightly more intelligent perl script "pdiff" is in ***pop/tools/validation***. "pdiff" allows you to set relative numerical differences which should be ignored.

Description of test problem

Domain? Boundary conditions
Flat bottom, open ocean, no land masses
Forcing: windstress and thermohaline
Parameter settings (see section on Options below)
Sample output files, named pop_*mode*.out, where *mode* is the value of the environment variable "ARCHDIR".

Making POP available for other users

The executable "pop" can be run from anywhere. All a user needs is a local working directory for data files. See the User Guide for details.

Guidelines for Unsupported Systems

Porting POP to an unsupported system

If your system is currently unsupported, we want to help you get it running anyway. When you do get it running, please tell us how you did it so we can incorporate that information in future versions of this document.

Where to get help

CSM-POP users

If you are using POP as a component of the NCAR CSM, technical support is provided by Matthew Hecht (hecht@ncar.ucar.edu, 303-497-xxxx).

LANL-POP users

If you obtained POP directly from LANL and are involved with its development, technical support will be provided through John Davis (jfd@lanl.gov, 505-667-4793) and Phil Jones (pwjones@lanl.gov, 505-667-6386).

Setup procedure

Extracting source code

It is a simple procedure to get and extract the POP source files. Go to the directory where you want to store the POP tar file. Get the compressed tar file, POP_nnn.tar.zzz from the FTP server. If the file name ends with 'gz', uncompress it with a 'zip' utility (eg., 'gunzip' on Unix or Linux). If it ends with 'Z', use the Unix standard 'uncompress'. You will end up with POP_nnn.tar. As you can verify with "tar tf POP_nnn.tar", the paths within the tarfile are relative, each beginning with "./".
Find a file system with at least ten megabytes free and create a working directory named anything you like, ***poptop*** for example.
Change directory to ***poptop*** and type "tar xf <path-of-**POP_***nnn*.tar>". In a short time you will have a tree something like this:

```
.../poptop/
       doc/
       setup_run_dir*
       setup_test_dir*
       mpi/
       serial/
       shmem/
       source/
       tools/
```

Setting up a working directory

A working directory is created by the C shell script setup_run_dir . You invoke it at the same directory level by typing

setup_run_dir directory [model]

where

directory is an arbitrary working directory name,
model is the kind of run for which you want to build POP..

(The default model is test; other possibilities are sector and 192x128x32.)

Setting environment variables

What happens in any working directory is determined by a single environment variable, ARCHDIR. Primarily, ARCHDIR sets the configuration for which POP is built. The configuration has two parts, the first for the ARCHitecture (sgi, unicos, linux, etc.), the second for the DIRectory (mpi, serial, shmem) containing code variants.
For example, ARCHDIR=sgi_mpi means that pop will run under Irix in parallel mode using MPI . ARCHDIR=sgi_serial means that pop will run under Irix on a single SGI processor in serial mode.

Parallel/vector/cache architecture options

Total number of processors

Pick an even number. The number depends on the available hardware arrangement.

X-Y assignment of processors

Factor the total number of processors into NPROC_X and NPROC_Y for the X and Y dimensions respectively. Edit the definitions in the working directory copy of ARCHDIR.gnu.

Configuring the make system for your machine

The following files are under every working directory as part of the make system. They are automatically invoked as needed by the gmake command.

fdepends.awk - *awk file for defining dependencies of .f files on other .f files containing modules.*

GNUmakefile - *gmake top-level control file which 'includes' or references others in this list.*

sgi_serial.gnu - *'included' when building a one-processor version of POP on Bluemountain.*

sgi_mpi.gnu - *'included' when building an MPI multiprocessor version on Bluemountain.*

sgi_shmem.gnu - *'included' when building a shared memory multiprocessor version on Bluemountain.*

t3d_serial.gnu - *'included' when building a serialr version on a Cray T3D or T3E.*

utilities.gnu - *'included' to define target "clean".*

compile/GNUmakefile - *referenced by level above. Originally designed to allow building while in the compile subdirectory, but that capability is no longer supported. (It still works, but it may go away in future versions.)*

Setting up for a build the first time

POP comes with a test problem built in. You should run it to see if POP will work correctly on your machine. Since your architecture does not match one of those currently supported, you will need to create a new configuration file and a new architecture-specific directory.

Let's walk through an example. Suppose you are porting to a cluster of HP workstations which has MPI already installed. You've untarred the POP distribution under a directory called ***pop***.

Create a .gnu file for your special configuration

Change directory to ***pop/input_templates.*** If you list it, the contents will look something like this:

```
CVS/                        model_size.F.test          sample_tavg_contents
GNUmakefile                 pop_in.192x128x32          sample_transport_file
GNUmakefile.compile         pop_in.256x128x20          sgi_mpi.gnu
fdepends.awk                pop_in.sector              sgi_serial.gnu
linux_mpi.gnu               pop_in.test                sgi_shmem.gnu
model_size.F.192x128x32     pop_sgi.log.test           t3d_serial.gnu
model_size.F.256x128x20     sample_history_contents    utilities.gnu
model_size.F.sector         sample_movie_contents
```

Look at the *.gnu* files. You do not see a file named *hp_mpi.gnu*, so you have to create one. Since you expect MPI to be fairly generic, make a copy of the existing file *sgi_mpi.gnu*.

Bring up *hp_sgi.gnu* in your favorite editor. Before you forget, change the title on line 2 from "sgi_mpi.gnu" to "hp_mpi.gnu".

The first major section has the definitions for FC (Fortran Compiler), LD (LoaDer), CC (C Compiler), etc. Some of these have only executable names, some have full paths. It is probably safest to determine exactly which executables you want, then put their full paths in the definitions.

ABI is a compiler/loader flag which sets the 64-bit processing option on SGI machines. You will probably need a different form.

Since you are going to use MPI, leave the definition of that symbol as is, "MPI = yes". Also leave TRAP_FPE as is, "= no" or delete the entire line.

The next section has the precompiler options. NPROCS, the total number of processors, is in effect defined as the product of NPROC_X and NPROC_Y. Assuming you will run with 4 processors, leave these both defined as "=2". Leave IVM defined as impvmix and leave COUPL undefined.

The next three sections set flags for C and Fortran compilers, loading and for libraries. The SGI flags will probably not be correct for your compilers.

Skipping some empty sections, you will come to "Implicit Rules for Compilation". The first thing it does is cancel any pre-existing rules for C and Fortran. Then it redefines those rules. If you want to understand the hieroglyphics, consult the Gmake manual. All you need to change is "SGI" to "HP" in the two echo lines. If you don't care to see how a make is progressing, you can comment out the lines beginning with "@echo".

Further on down, you will see a section "Implicit Rules for Dependencies". All you need to change here is "SGI" to "HP" in four echo lines.

Create a working directory for a test problem

You need a clean working directory directly under *pop.* Pick a name that doesn't already exist, say *trial*. With your default directory as *pop,* type "*setup_run_dir trial test*". You will see a screenful of informative messages ending with "Copying ../trial/compile/GNUmakefile".

Building POP normally

Make sure your environment variable ARCHDIR is set to "hp_mpi". Change to the *trial* directory you just created. Type "*gmake*". Wait. When gmake finishes, your working directory *trial* will have an executable file named *pop.* You are ready to run.

Building a debug version

Debugging a parallel code is not always easy and is probably not the best way to solve portability problems with POP. Inserting print statements into the source code and recompiling is usually faster. But if you must, here is how to do it.

To build a debug version, ***pop_db***, type "gmake OPTIMIZE=no". ("gmake" by itself will build the optimized executable ***pop***.) Object files are no longer archived in a separate library, so it is necessary to do "gmake clean" when switching between optimized and non-optimized versions.

Running the test problem to verify the port

type "mpirun –np 4 pop > *somefilename.out*". It should finish in a minute or so. Compare *somefilename.out* with the read-only sample output file "pop_sgi.log" in the same directory. Most systems will have "diff" and "xdiff" available for comparing these text files. A slightly more intelligent perl script "pdiff" is in ***pop/tools/validation***. "pdiff" allows you to set relative numerical differences which should be ignored.

Description of test problem

Domain? Boundary conditions
Flat bottom, open ocean, no land masses
Forcing: windstress and thermohaline
Parameter settings (see section on Options below)
Sample output files, named pop_*mode*.out, where *mode* is the value of the environment variable "ARCHDIR".