Code structure of Enviro-HIRLAM

Ulrik Smith Korsholm
usn@dmi.dk
Research Department
Danish Meteorological Institute

Code structure

hirlam_src : original source code (reference HIRLAM);
do not make any changes in this directory !

dmiref_src : local changes to the reference version;
this is where all modifications are done

Sub-directories: grdy (dynamics routines), phys (physics routines),
gcod (GRIB encoding/decoding routines), ...

Path to sub-directories: ~USER/compile dir/dmiref_src/hirlam/

directory of your choice

from here the directory structure is locked

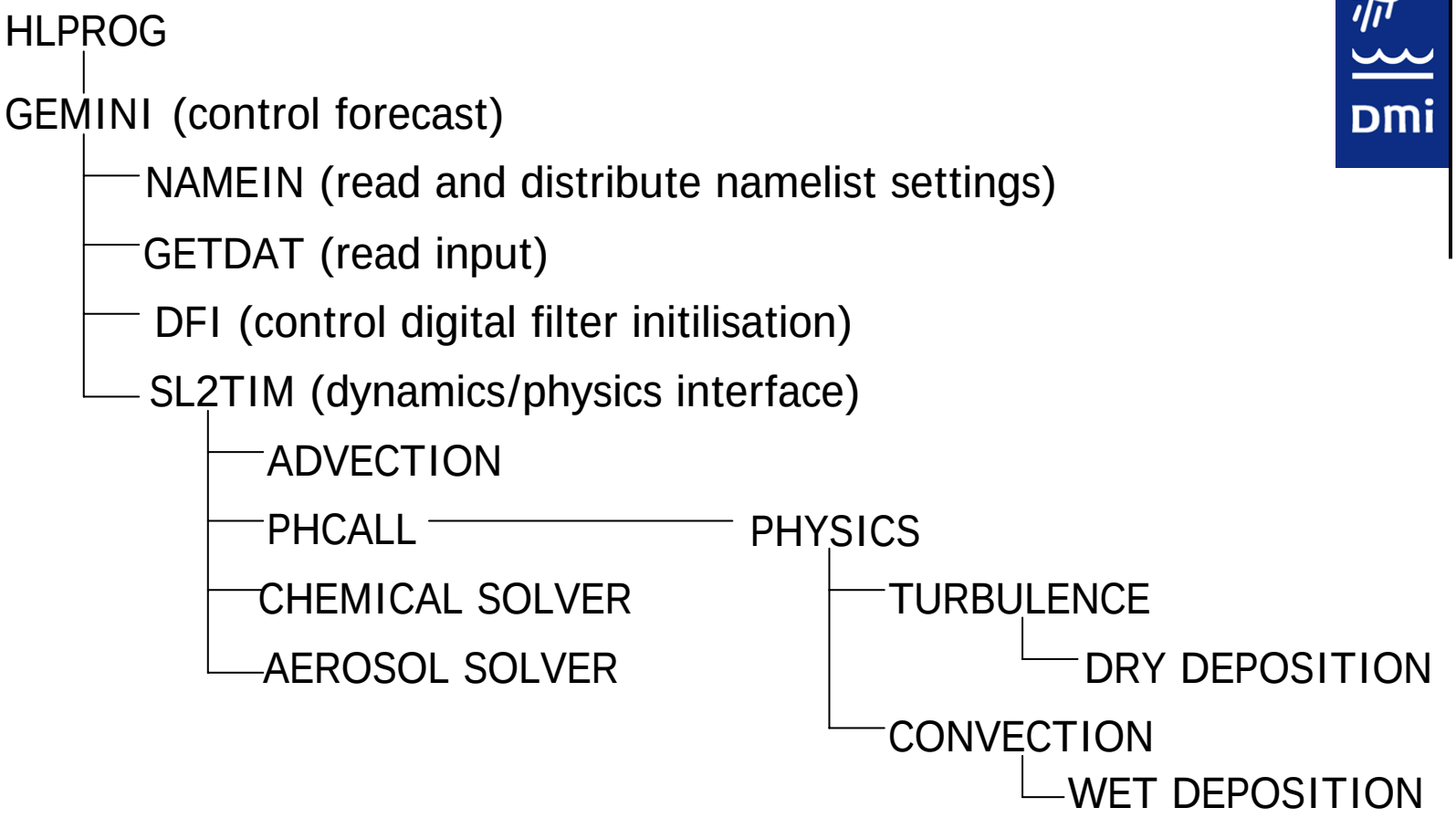
Source mainly F77, some F90 and very little C

All new features should be in F90 using
the modularity paradigm

Hence, make (if possible) self contained modules,
with your sub-routines, definitions, etc.

WARNING: BE AWARE OF COMPILATION PROBLEMS
WHEN MIXING F77 AND F90

Overview call tree for the environmental part:



Compilation

Topics: How to compile, compiler options,
how to add a new sub-routine

Compilation scripts based on PEARL and MAKEFILE

Compile from dmiref_src using intel80.sh
Modify to give path to hirlam_src

Compilation

Compile script intel80.sh

```
#!/bin/tcsh

ARCH=intel80
WORKDIR=$PWD
HIRLAMVERSION=current
CVSDIR=:pserver:anonymous@cvs:2401/data/cvsroot
CVSREPOSITORY=hirlam_src
SCRATCHDIR=/data/usn/projects/enviro-hirlam/src/chemistry
MPPMAKE="-j 1"
HGSOPT=nohgs

export ARCH WORKDIR HIRLAMVERSION CVSDIR CVSREPOSITORY
SCRATCHDIR MPPMAKE HGSOPT

perl Compile_HIRLAM.pl hlprog.x
exit
```

Bdprep, allclean

Compilation

Configuration files: config.”architecture” Snapshot

```
ARCH=intel80fftw3
# Arch part:
MACHINECPP= -DFUJITSU -DLINUXPGI -DLITTLE_ENDIAN -DPREC32 -DWORDRECLEN -DGRIB32
-DINTEL80
MACHINECPPHIRLAM=$(MACHINECPP)
CRAYDEF=-DNONCRAYF
CPP = /lib/cpp -E -traditional
CC = gcc
CCFLAGS = -c -DFUJITSU -DPREC32 -I$(ROOTDIR)/ifsaux/ioassign/
-I$(ROOTDIR)/ifsaux/pcma/ -DHIRLAM
FC = ifort
FCFLAGS_CMA = -real_size 64 -O2 -fixed -c
FCFLAGS_CMA90 = -real_size 64 -O2 -free -c -I$(ROOTDIR)/$(ARCH)/cmamod
MODEXT=mod
FCFLAGS_VAR = -real_size 64 -O2 -fixed -c -I$(ROOTDIR)/$(ARCH)/varmod
FCFLAGS_VAR90 = -real_size 64 -O2 -free -c -I$(ROOTDIR)/$(ARCH)/varmod
FCFLAGS_HIRLAM = -fixed -O2 -c
LD = ifort
LD_HIRLAM = ifort
LD_MPP = ifort
LD_MPP_HIRLAM  = ifort
LDFLAGS_VAR = -Vaxlib -nothreads /home/ksm/fftw/lib/libfftw3.a -o
LDFLAGS_CMA = -Vaxlib -nothreads -o
LDFLAGS_HIRLAM  = -Vaxlib -nothreads -nofort_main -o
```

Compilation flags
you can define your own

FORTRAN compiler

Compilation options

Linker

Linker flags, libraries

Compilation

Note: You can define your own configuration files

Main compilation script ”Compile_HIRLAM.pl” invoked from Intel.”architecture”.sh

- Set-up directory structure
- Is code new or old ?
- Enters all subdirectories and executes individual make files
- Executes overall makefile and link all archives together

Location of compiled code (pre-processed files, object files):
~hirlam_src/”architecture”/hirlam/”sub-dirs”

Note: Compilation and code error messages refer to the compiled code, not the original source !!!

Compilation

Locaton of executables: ~hirlamsrc/"architecture"/bin

Executables:

hlprog.x : main Enviro-HIRLAM executable
interp.x : horizontal interpolation
vineta.x : vertical interpolation
bdprep.x : boundary preparation
span.x : surface data assimilation

The rest are related to upper air data assimilation

How to make changes

Changes can be made directly in code: easy, the compiler will recognize your changes

Subroutines may be added in sub-directories: if it is in its own file add the file name to the local makefile and dependencies.inc

New directories may be added along side grdy etc.: must make changes in compile scripts; not recommended; much easier in the new version.

In general always follow the code structure which already exists

Optimization

Main optimization issues

- Parallelization
- Compiler optimization
- I/O optimization; HGS
- Other fixes (Helmholtz, loop length)

Optimization

Optimization strategy dependent on computer architecure:

Cash vs. no cash

Vector vs. Scalar (loop lengths, compiler options)

Shared memory vs. Distributed (parallelization strategy)

Note: current version optimized for vector machines (NEC-SX6); New version will be optimized for scalar machines (CRAY-XT5)

NEC-SX6
Theoretical peak:
0.5 Tflops

CRAY XT-5
Theoretical peak:
35 Tflops

Optimization

Parallelization strategy: single program multiple data paradigm:

- Horizontal grid divided into sub-domains (specified by user; decompose)
- MPI is used for inter-process communication
- Each MPI task is associated with one cpu and holds one sub-domain
- Each task performs exactly the same calculations but on different parts of the grid
- Task with rank 0 is master all others are slaves
- Halo hard coded to 2 grid points
- Extra points for semi-Lagrangian departure point calculation is also called halo in the code, be careful not to mix them up
- Physics and chemistry parts are embarrisingly parallel

Optimization

Grid decomposition options are defined in run script:
NPROCX: number of sub-domains in longitudinal direction

A symmetric decomposition will generally give the most even workload
And thereby use the processes optimally -> fastest execution time

Do not compromise the parallelization strategy: often very difficult to locate and solve parallelization errors

Note: do not introduce horizontal dependencies beyond the halo

Optimization

Compiler optimization

Compilers often very effective in optimizing code to a particular Architecture; but sometimes it needs help.

When you implement something be sure to do timings and determine whether you destroy optimization.

Maybe you just prevented the compiler optimization.

Be sure to get maximum info from the compiler, often it wil state if it cannot optimize e.g. a given loop.

Often optimization is invoked by –Ox where x is the degree of optimization

Play around with such options to get the optimal settings for your computer architecture

Optimization

I/O optimization

All I/O normally done synchroneously on master process
All other processes wait untill reading/writing is done before they proceed
HGS: HIRLAM Grib File Server

Output is written to memory asyncroneously (much faster than writing to disk) by each task, which then proceeds with calculations.

A few tasks handles the memory queues and actual writing

If input is needed it is requested in due time; when the request is processed by HGS it is put in memory and used when needed

Based on F90 and MPI; NOT TESTED WITH THE ENVIRO PART BUT IS AVAILABLE; expected gain in runtime: more than 17 %.

Optimization

Other optimizations:

FFT's and Gaussian elimination is used for the solution of the Helmholtz equation.
It pays off to switch the lat and lon indices before FFT, i.e grid is rotated Before the FFT and inverse FFT's are performed.

Loop length is currently set to about 1300, due to vectorization. This Should be adjusted to your computer architecture.

Script system

¨Located in a workng directory of your choice in HLM directory

Controls execution of model/cycles e.g.

- linking of input files,
- generation of boundaries,
- options which are external to the model,
- generation of namelists containing options which are internal to the model, ...

Script system

Based on PEARL modules:

HLPROG.pm -> options and namelists for a forecast;
from here the executable is invoked
GD.pm -> Contains all controls paramters of a
forecast; most re set in the run script,
but some are to be set in GD.pm.
Determines how cycles are executed
Bndprep.pm -> Controls preparation of boundary files
Single.pm -> Overall control of cycles, e.g. Bndprep,
data assimilation, forecast, archiving, ...

To add new namelist parameter you must add it in HLPROG.pm and do some source code changes so it is read properly.
NOTE: Always remember memory allocation issues when adding namelists

I/O

Input data

Boundary files: interpolated from an outer model,
typically every 3 hours,
meteorological + tracer fields interpolated,
may be generated if input data exists,
file names: b$gridname$date+$hour

Initial values : analysis based on first guess from previous run,
cold start using 00 boundary file,
must exist,
file names: hi$gridname$date_time+$hour

First guess : output from previous run is needed for surface
and upper air analysis

Climate data, surface observations, upper air observations

I/O

Output data

Model level files: selected prognostic and dioagnostic fields on model levels,
file name: hi$gridname$date_time+$hour,
typical size: 300 Mb !! Depends on number of points !!

Pressure level files: selected fields interepolated to pressure levels: 1000 mb,
850 mb, 800 mb, 700 mb, 500 mb, 250 mb,
file name: ba$date_time+$hour.p
typical size: small

Surface files: selected fields interpolated to surface,
file name: ba$date_time+$hourul
typical size: small

I/O

Boundary file generation

Use input from an outer model which contains the modelling domain

Remember that input must have the correct valid time !!!

The script system will generate the appropriate boundaries, from the ”bdprep” executables if it is told to do it.

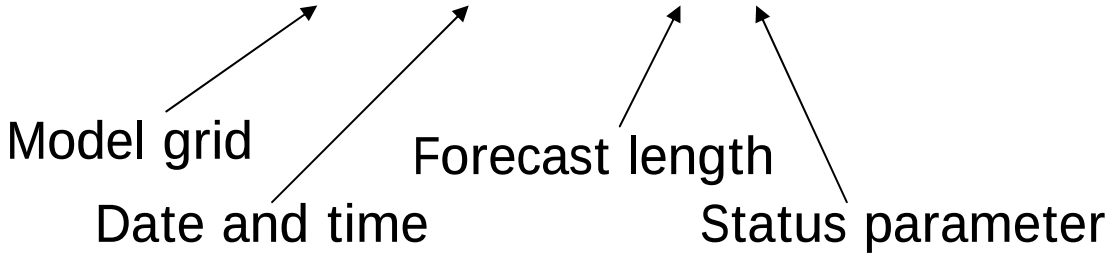
How to run the model

Choose working output directories

Make a temorary directory ”/tmp” with path: ~working dir/tmp

All other directories will be generated upon execution

/tmp should contain the ”.unique” file: T15 08102312 48 3



Status 0: start all over; generate boundaries, execute surface and upper air data assimilation

Status 3: go directly to the forecast, assuming all input data is available

NOTE: Info in .unigue must match info in the runscript

How to run the model

Run script (PEARL) ⟷ Script system (PEARL) ⟷ Enviro HIRLAM

The run script contains the options which are most often varied, such as the length of the time step, name of grid, output frequency, etc.

Also contains parallelisation and queing system options

This is the script you execute in order to run the model

How to run the model

Part of Runscript

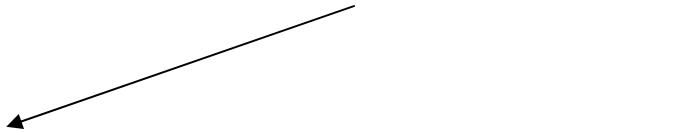
Here you put queing system Commands, if there is one

Specify working directory and executables

```
#!/usr/bin/perl

# RUN SCRIPT FOR OPERATIONAL REF-HIRLAM RUNS.

BEGIN {

#------------------------------------------------------------
# Define paths
# ------------------------------------------------------------
 my($workdir)="/data/usn/projects/enviro-hirlam/scr/tracer_enviro/"; # Location
of script system.
 my($tmpdir)=$workdir.'tmp/'; # Location of temporary files
 my($temp)=$tmpdir.'tmp.ref'; # Extra temporary directory used during execution
# $ENV{CONST}="/sx6opr/hiropr/fm/refhirlam/data/dataopr/"; # Location of
constant files: binaries, climate and orography.
 $ENV{CONST}="/xtmp/usn/source/tracer/testetex/";
 my($exedir)="/data/usn/projects/enviro-hirlam/src/chemistry/hirlam_src/intel80/bin/";
```

How to run the model

Part of Runscript




```
$ENV{BOUND} =$tmpdir."bound/";
 $ENV{BDBDIR}=$tmpdir."bdb/"; # Location of buffer data
 $ENV{ECSSTDIR}=$tmpdir."ecsst/"; # Location of sst data
 $ENV{SPOOLDIR}=$tmpdir."spool"; # Storage of output DMI-GRIB files
 $ENV{INPUTDIR}=$tmpdir."gdb";
 $ENV{OUTPUTDIR}=$tmpdir."gdb/";
 $ENV{GDBOPR94}=$tmpdir."spoola";
```

How to run the model

Part of Runscript

```
my $ntaskperhost = 1; # Mpi tasks used in forecast
my $omp          = 1; # OpenMP threads used in forecast
my $nhosts       = 1; # Number of nodes used in forecast
my $nprocx       = 1; # Sub-domains in longitudinal direction
my $nproc_hgs    = 0; # Number of dedicated output servers
my $iniopt       = "noini"; # Initilisation scheme: noini, dfiini or nmiini
my $output       = "yes"; # Write out output (yes) or not (no)
my $outfrq       = 1; # Output frequency (hours - 1,3,6,...)
my $ndtimGsl     = 600; # Semi-lagrangian time step for G model in sec.
my $ndtimDsl     = 150; # Semi-lagrangian time step for D model in sec.

------
$w->{period}->{begin}=dateformat($tref-1*86400);
$w->{period}->{begin}=1994102312; #Start date and time for historical period
$w->{period}->{end}=1994102312;
```

How to run the model

For plotting of GRIB files use either METGRAF or GRADS (see GRADS webpage)