Debugging

Institute of Meteorology and Climatology, Leibniz Universität Hannover

Principal sources of errors

PALM runs can give rise to a large variety of errors ...

Some of the main possible reasons for errors are:

- Missing or wrong options in the **`palmrun`** call.
- Errors in the configuration files.
- Errors in the NAMELIST parameter file.
- Errors in the `ssh`-installation (authentication), if a remote host is used for batch jobs.
- FORTRAN errors in the user code (user-interface files).
- PALM runtime errors due to:
  - wrong parameter settings,
  - errors in the user code,
  - errors in PALM‘s default code, which have not been detected so far (e.g. because some parameter combinations have never been tried so far).

First steps of debugging

Find out the principal reason of the error(s):

- Carefully analyze the job protocol file (or messages on the terminal, in case of interactive runs) for any error messages or unexpected behaviour.
- In case of batch runs on a remote host, if the job protocol file is missing on the local host, try if you can find it in **`~/job_queue`** on the remote host.
- If the job has run into a time limit, no job protocol files or messages might be created at all (system depending).
- Some typical errors which may occur during execution of **`palmrun`** are automatically detected and displayed by **`palmrun`** in the job protocol or on the terminal: Respective error messages will begin with "**+++**".
- Compile and run time error messages will only appear in the job protocol or on the terminal (in case of interactive runs).
- In case of runtime errors, terminal messages may give first helpful hints about where errors are located.
- Check the trouble tickets for possible solutions, in case that someone ran into a similar problem. (https://palm-model.org/trac/wiki/tickets)

Debugging runtime errors (I)

- In case of runtime errors, the available information depends on the compiler and on the compiler settings.
- The default options for the Intel-compiler (**`-O3`** for fast execution) give almost no information, e.g. about the subroutine or the line number of the code where the error occured. Execution is even continued in case of floating point errors!
- Floating point error detection and traceback can be activated with compiler options

```
ifort -fpe0 -debug -traceback -O0 ...
```

- A default **`.palm.config.default_trace`** for the trace mode can be found in folder **`palm_model_system/packages/palm/share/config`**.
- Settings for **`default`**:

```
%compiler_options    –fpe0 –O3 –fp-model source
```

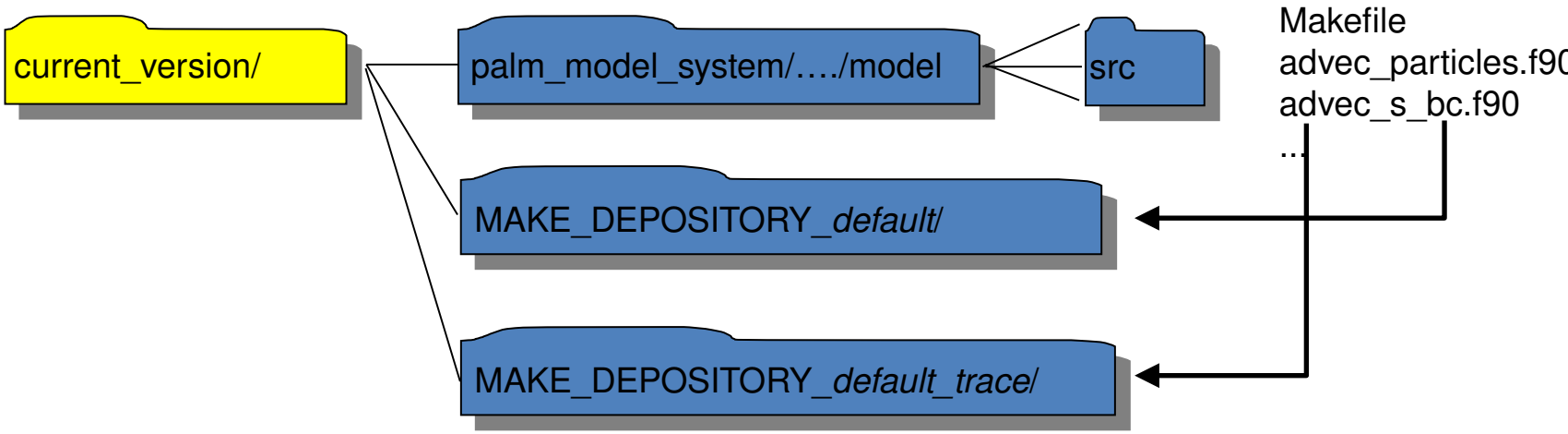
- Settings for **`default_trace`**:

```
%compiler_options  -fpe0 –O0 -check –traceback –g …
```

- See https://www.palm-model.org/trac/wiki/doc/app/recommended_compiler_options for debug options required by other compilers.

Debugging runtime errors (II)

- Calling **`palmbuild -c default_trace`** will compile the code with compiler options provided by **`.palm.config.default_trace`**. The pre-compiled code will be put into a separate make depository:



- The **`palmrun`** option **`-c`** (the configuration identifier) decides, which version is used:

  `palmrun ... -c default ...` will use the optimized version
  `palmrun ... -c default_trace ...` will use the debug version

Enabling debug options slows down the execution speed significantly!

Debugging runtime errors (III)

- Still often these options do not help to determine and localize the problem.
- There are several ways of handling these cases:
  - PALM standard location messages which appear on the terminal (interactive mode) or in the job protocol (batch mode).
  - PALM standard debug messages which are output into specific debug files. The messages explicitly need to be switched on.
  - The print/write debugger: Manual output of additional informations to the debug files.
  - debuggers like **`dbx`** or GUI-based debuggers like **`“totalview“`** or **`“Allinea DDT“`**

PALM location messages

- Printed to terminal or job protocol:

```
*** execution starts in directory
     "/localdata/raasch/example_cbl.766"
------------------------------------------------------------------------------
*** running on: bora bora bora bora
*** execute command:
    "mpiexec -machinefile hostfile -n 4 palm"

 15:23:18   -start----  reading environment parameters from ENVPAR
 15:23:18   -finished-  reading environment parameters from ENVPAR
 15:23:18   -start----  reading NAMELIST parameters from PARIN
 15:23:18   -finished-  reading NAMELIST parameters from PARIN
 15:23:18   -start----  creating virtual PE grids + MPI derived data types
 15:23:18   -finished-  creating virtual PE grids + MPI derived data types
 15:23:18   -start----  checking parameters
 15:23:18   -finished-  checking parameters
 15:23:18   -start----  model initialization
 15:23:18   -start----  initializing surface layer
 15:23:18   -finished-  initializing surface layer
 15:23:18   -finished-  model initialization
 15:23:18   -start----  atmosphere (and/or ocean) time-stepping
```

- In the PALM code, e.g. SOURCE file **`check_parameters.f90`**:

```
CALL location_message( 'checking parameters', 'start' )
...
CALL location_message( 'checking parameters', 'finished' )
```

PALM debug messages (I)

- Output of debug messages requires setting of runtime parameters

  `debug_output = .TRUE., debug_output_timestep = .TRUE.`

- Output will be written into files **`DEBUG_000000`**, **`DEBUG_000001`**, etc. in PALM's temporary working directory. You need to set **`palmrun`**-option "**`-B`**", because otherwise the temporary working directory is deleted at the end of the run!
- The name of PALM's temporary working directory is generated from environment variable **`fast_io_catalog`** (see **`.palm.config.<ci>`**), the run identifier, and a five digit random number:

  `/<fast_io_catalog>/<run identifier>.<random number>`

- Contents of a debug file look like this:

```
System time: 09:27:25 | simulated time (s): 0.000 |  -start- reading module-specific parameters
System time: 09:27:25 | simulated time (s): 0.000 |  -end--- reading module-specific parameters
System time: 09:27:26 | simulated time (s): 0.000 |  -start- checking module-specific data output ts
System time: 09:27:26 | simulated time (s): 0.000 |  -end--- checking module-specific data output ts
System time: 09:27:26 | simulated time (s): 0.000 |  -start- checking module-specific parameters
System time: 09:27:26 | simulated time (s): 0.000 |  -end--- checking module-specific parameters
System time: 09:27:26 | simulated time (s): 0.000 |  -start- allocating arrays
System time: 09:27:26 | simulated time (s): 0.000 |  -start- initializing module-specific arrays
```

- In the PALM code, e.g. the source file **`init_3d_model.f90`**:

```
IF ( debug_output )  CALL debug_message( 'allocating arrays', 'start' )
...
IF ( debug_output )  CALL debug_message( 'allocating arrays', 'end' )
```

PALM debug messages (II)

- If required, add additional messages in the default code (or your user-interface code) to narrow down the location.

```
IF ( debug_output )  CALL debug_message( 'user position 1', 'start' )
```

- After the location has been identified, you can write values of specific variables to the debug file (which has FORTRAN I/O unit number 9), which you suspect to cause the problem:

```
IF ( debug_output )  THEN
   CALL debug_message( 'user position 1', 'start' )
   WRITE( 9, * ) 'variable a = ', a, 'variable b = ', b
   FLUSH( 9 )
ENDIF
```

- **Very important:** All output is buffered, i.e. it will not be directly written to disc. In case of program aborts, the buffer contents are lost, so the output of the last write statements are not available. You have to prevent this problem by flushing the buffer after **each** print/write statement.
- Don‘t forget to re-compile with **`palmbuild`** after you have modified the code!