Compiling Environment – ecgate

Xavi Abellan

xavier.abellan@ecmwf.int

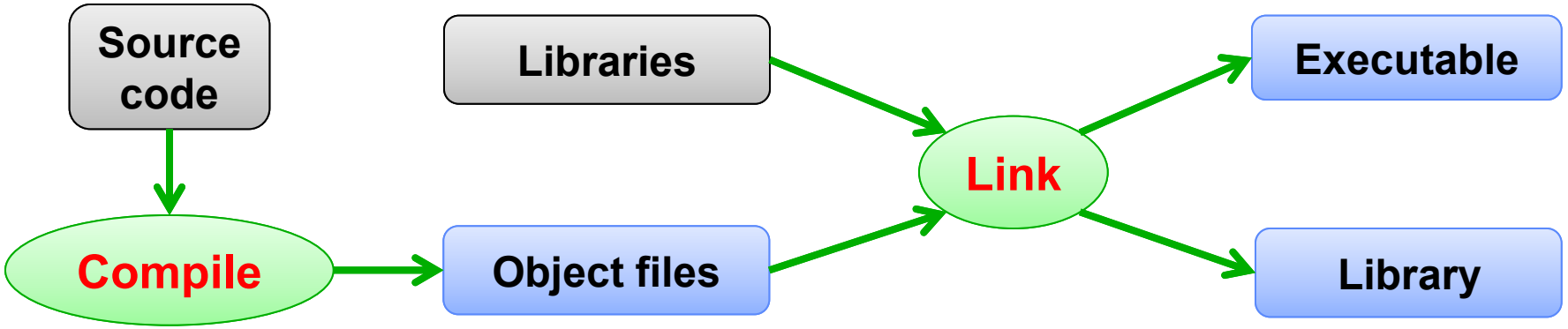
User Support

Content

- Introduction
- Fortran Compiler
- Linking
- Libraries
- Make
- Debugging
- Profiling
- Practical session

Introduction

- Compiling
  - Objects
- Linking
  - Libraries
    - static libraries
    - shared libraries



Introduction

- Why compiling at ECMWF?
  - decoding of (MARS) data
  - model runs
- Alternatives to compilation?
  - Grib_api tools, grib_api python interface
  - Netcdf format generated from MARS
  - Wgrib, cdo, …
- Which platforms are available?
  - Linux server (ecgate)
  - Supercomputers (IBM: c2a – Cray: cca)

Introduction

- Which compilers?
  - Fortran (77/90/95/2003)
  - C/C++
- Which platform to use?
  - High Performance Computing Facility (c2a or cca) for computing intensive work, including any // work.
  - Linux server (ecgate) for decoding or I/O bound work.

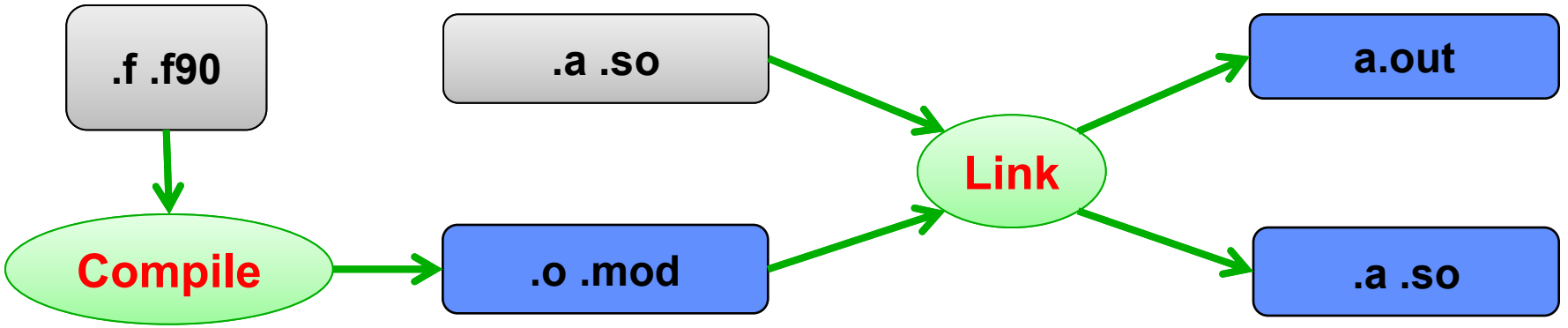
Compilers

- GNU compilers:
  - gfortran
  - gcc
  - g++
- Which version do I use?

```
$ gfortran --version      # currently version 4.4.7
$ gcc --version           # currently version  4.4.7
```

Compilers

- Common file suffixes and files
  - .f, .F, .f90, .F90 : source code
  - .o : object file
  - .a : archive file (library)
  - .so: share object (library)
  - .mod: fortran 90 module files
  - a.out: default name of executable



Fortran Compiler common options

- Fortran 77 / f90
  - -c compilation only, no linking
  - -fdefault-real-8 64bit real variables
  - -O[1-3] optimisation
  - -g debugging
  - -v verbose
  - --help display usage
- Many more options. See man page.

Compilation – return codes

- Return code
  - Successful compilation: 0
  - Failure: (≠0) 1, …
  - gfortran messages:

```
demo.f90:3.18:

    use grib_api
                 1
Fatal Error: Can't open module file 'grib_api.mod' for reading at (1): No
such file or directory
```

position

Error severity

Error message

Word lengths – precision

- 32bit real and integer variables by default.
- The option –fdefault-real-8 promotes real variables to 64bit entities.
- When using a library, check its precision, e.g. for EMOSLIB, MAGICS. The GRIB_API is independent of the precision for floating points.

Fortran I/O

- GRIB and BUFR formats are pure binary formats, accessible with PBIO routines from EMOSLIB or with the GRIB API for GRIB (Edition 1 and 2).
- IEEE format - big-endian on c2a, little endian on Linux systems (ecgate and Cray)
  - real*4: 6 significant digits
  - real*8: 15 significant digits
  - Use '-fconvert=big-endian' to read/write big-endian files.

Linking

- Use gfortran to link, e.g.

```
$ gfortran -o prog prog.f $EMOSLIB
# equivalent to:
$ gfortran -o prog prog.f –L/usr/local/apps/libemos/000393/lib \
    –lemos.R32.D64.I32
```

Beware of duplicated entries.

- Use "ar" to build static libraries, eg.

```
$ gfortran –c *.f
$ ar -vr libmy.a *.o
```

Libraries

- ECMWF libraries
  - Graphics software library – MAGICS: $MAGPLUSLIB_STATIC (Magics++)
  - Meteorological Software - EMOS library - $EMOSLIB
  - Grib_api, for GRIB1 and GRIB2 format - $GRIB_API_LIB, $GRIB_API_INCLUDE
  - Locally produced software library - EC Library - $ECLIB
- Manufacturer/Public Domain Libraries
  - BLAS/LAPACK – public domain software
  - HDF/NetCDF available. See

    www.ecmwf.int/services/computing/docs/data_formats/HDF_netCDF.html

Libraries (cont)

- Many of our locally produced libraries have both 32-bit and 64-bit floating point versions (REAL numbers) - different libraries.
- Do NOT make the confusion between the precision (32/64 bit REALS) and the ADDRESSING mode (32/64 bit) of a library:
  - You will get WRONG results when mixing libraries of different precision.
  - You will not be able to link your program if you mix libraries of different addressing mode.

Make

- Easy to use utility to build a program or library.
- Suitable for different languages.
- Makefile: file containing rules on how to compile code and build library or executable.
- The ‘make’ command will read the Makefile and will figure out which code files (or libraries or executables) need to be rebuilt.
- make allows for compilations in parallel (make –j).

Makefiles

- Contain rules that will be applied in cascade:
- The command(s) to run for each rule must be preceded by a tab
  - No spaces!!!

- Syntax:

```
target1: source1
        command_to_run target1 source1
```

- Example:

```
hello:   hello.f
        $(FC) -o $@ -ffixed-form $(FFLAGS) $<
```

Debugging

- checking:
  - array bounds checking: -fbounds-check

    $ gfortran –fbound-check prog.f –o prog

    $ ./prog

    checking done at runtime
  - undefined reference checking

    $ gfortran –finit-real=inf prog.f –o prog

    checking done at runtime
- generating debug output:
  - Backtrace: -fbacktrace
  - Core file: -fdump-core

Debugging – floating point exceptions

- nothing generated on floating point exception.
- Floating point trapping
  - $ gfortran –ffpe-trap=overflow,invalid,zero [-g] [-O0] prog.f –o prog
  - $ ./prog
  - ...
- interactive window based debugger: - totalview
  - $ module load totalview
  - $ totalview ./prog
- Core files – how to get a backtrace
  - $ gdb –c core ./prog
  - ➢ where

Profiling - tuning

- time - command timer

  $ time a.out

- -O and other options at compilation for faster execution. Try to use –O3

- other applications, like gprof

  $ gfortran –O0 -g –pg –o prog prog.f

  $ ./prog

  $ gprof prog gmont.out

References

- GNU manuals (fortran, C, …):
  http://gcc.gnu.org/onlinedocs/
- ECMWF fortran pages:
  www.ecmwf.int/services/computing/docs/fortran/
- ECMWF computing pages:
  www.ecmwf.int/services/computing/
- Job examples:
  www.ecmwf.int/services/computing/job_examples/ecgate/