Experiments with CMake

Yurii Batrak (MET-Norway)

II·APR·MMXIX

TL;DR: using CMake could considerably reduce build time

	Makeup	CMake	
		make	Ninja
Configuration	00:10:00	00:01:30	00:01:40
Full build	01:15:00	00:20:00	00:18:27
No-op build	00:06:50	00:00:40	00:01:35
Incremental build	00:07:00	00:01:20	00:02:22

Why should we care about the build system?

Makeup does a very good job compiling our system...

Why should we care about the build system?

Makeup does a very good job compiling our system...

BUT

Why should we care about the build system?

Makeup does a very good job compiling our system...

BUT

Research experiments often involve a “rapid refresh” approach

It takes a lot of time to recompile the system after changing a single source file

Why should we care about the build system?

Makeup does a very good job compiling our system...

BUT

Research experiments often involve a “rapid refresh” approach

It takes a lot of time to recompile the system after changing a single source file

THERE SHOULD BE A BETTER WAY!

Problems with Makeup-build

- In-house build system within a rather small community
  - No feedback or support from outside our community
  - Custom Perl-powered tools to configure the build
  - Could be difficult for newcomers

Problems with Makeup-build

- In-house build system within a rather small community
  - No feedback or support from outside our community
  - Custom Perl-powered tools to configure the build
  - Could be difficult for newcomers
- Long compile times both initial and incremental
  - Full build takes more than an hour
  - Add some minutes for `Makeup_configure`
  - No-op build is not really no-op

Problems with Makeup-build

- In-house build system within a rather small community
  - No feedback or support from outside our community
  - Custom Perl-powered tools to configure the build
  - Could be difficult for newcomers
- Long compile times both initial and incremental
  - Full build takes more than an hour
  - Add some minutes for `Makeup_configure`
  - No-op build is not really no-op
- Users should know “when” and remember “to” submit Configure job if sources are changed
  - For some edits you could skip `Makeup_configure`
  - But in some cases it is required to re-run `Makeup_configure`

Problems with Makeup-build

- In-house build system within a rather small community
  - No feedback or support from outside our community
  - Custom Perl-powered tools to configure the build
  - Could be difficult for newcomers
- Long compile times both initial and incremental
  - Full build takes more than an hour
  - Add some minutes for `Makeup_configure`
  - No-op build is not really no-op
- Users should know “when” and remember “to” submit Configure job if sources are changed
  - For some edits you could skip `Makeup_configure`
  - But in some cases it is required to re-run `Makeup_configure`
- Not all dependencies are expressed within the build system
  - Try to add Fortran flags to your `config.*`
  - Good luck with convincing Makeup to rebuild something...

Is it even possible to build ALADIN-HIRLAM NWP system with CMake?

CMake in a nutshell:

- Build system generator
- Open-source
- Widely adopted, large community
- Fortran is a “first-class” citizen

Features of our source code:

- A LOT of generated sources
- Strongly coupled components
- Mixture of Fortran and C code

The best project structure for CMake is a directed acyclic graph

So, how difficult is to write a CMake build?

Generation of interface blocks

- List of generated interfaces should be know at configure time
- Only the top-level subroutines and functions require interface blocks
- These routines could be found with a simple CMake script
- But it restricts the declarations to use named end-statements
- Added target should be listed as a dependency of the first library

```
1  set(all_ifaces "")
2  foreach(dir IN ITEMS
3    arpifs aladin ifsaux/ddh satrad)
4    hm_glob(Fortran
5      RECURSE
6      SOURCES src
7      DIRS ${dir})
8    hm_list_generated_interfaces(
9      QUIET
10     SILENT
11     SOURCES ${src}
12     INTERFACES ifaces)
13   list(APPEND all_ifaces "${ifaces}")
14 endforeach()
15
16 add_custom_target(
17   generate_interfaces
18   DEPENDS ${all_ifaces})
```

Generation of interface blocks

```
1  find_top_level_routines("foo.F90" top_level_routines ${ARG_QUIET})
2  # ... Invoke make_intfbl.pl ...
3  add_custom_command(
4    OUTPUT ${out_dir}/foo.intfb.h
5    COMMAND ${ENV} perl -I${makeup_dir} ${makeup_dir}/${generator} foo.F90
6
7    DEPENDS "foo.F90"
8    WORKING_DIRECTORY ${out_dir}
9
10   VERBATIM
11 )
12 list(APPEND all_generated_interfaces "${out_dir}/foo.intfb.h")
```

Naïve realisation that just calls `make_intfbl.pl` works, but...

Generation of interface blocks

```
1  find_top_level_routines("foo.F90" top_level_routines ${ARG_QUIET})
2  # ... Invoke make_intfbl.pl ...
3  add_custom_command(
4    OUTPUT ${out_dir}/foo.intfb.h
5    COMMAND ${ENV} perl -I${makeup_dir} ${makeup_dir}/${generator} foo.F90
6
7    DEPENDS "foo.F90"
8    WORKING_DIRECTORY ${out_dir}
9
10   VERBATIM
11 )
12 list(APPEND all_generated_interfaces "${out_dir}/foo.intfb.h")
```

Naïve realisation that just calls `make_intfbl.pl` works, but...
...`make_intfbl.pl` is smart enough to not touch unmodified interfaces.

It could trigger generation on each rebuild after modifying a source file.

Generation of interface blocks

```
1  find_top_level_routines("foo.F90" top_level_routines ${ARG_QUIET})
2  # ... Invoke make_intfbl.pl ...
3  add_custom_command(
4    OUTPUT ${out_dir}/foo.intfb.h.stamp
5    COMMAND ${ENV} perl -I${makeup_dir} ${makeup_dir}/${generator} foo.F90
6    COMMAND ${CMAKE_COMMAND} -E touch ${out_dir}/foo.intfb.h.stamp
7    DEPENDS "foo.F90"
8    WORKING_DIRECTORY ${out_dir}
9    BYPRODUCTS ${out_dir}/foo.intfb.h
10   VERBATIM
11 )
12 list(APPEND all_generated_interfaces "${out_dir}/foo.intfb.h")
```

Naïve realisation that just calls `make_intfbl.pl` works, but...
...`make_intfbl.pl` is smart enough to not touch unmodified interfaces.

It could trigger generation on each rebuild after modifying a source file.

This could be avoided by using stamp files

Generation of interface modules

- SURFEX uses generated interface modules instead of interface blocks
- Generated interface modules are prepared by a call to the same function
- But returned modules list is added directly to the list of SURFEX sources

```
hm_glob(Fortran
  SOURCES SURFEX_ALL_SRC
  DIRS
    ASSIM
    GELATO
    OFFLIN
    SURFEX
    TOPD
    TRIP)
```

```
hm_list_generated_interfaces(
  SURFEX
  QUIET
  SILENT
  SOURCES ${SURFEX_ALL_SRC}
  INTERFACES generated_interfaces)
list(APPEND SURFEX_ALL_SRC
  "${generated_interfaces}")
```

Code generation: blacklist

- Code generation for blacklist is done by a pre-built compiler tool
- Blacklist compiler build configuration is straightforward
- CMake recognizes it as a dependency of the blacklist object

```
1  bison_target(
2    blacklist_parser
3    compiler/yacc.y
4    ${...}/y.tab.c)
5  flex_target(
6    blacklist_lexer compiler/lex.l
7    ${...}/blacklist_lexer.c)
8  add_flex_bison_dependency(
9    blacklist_lexer
10   blacklist_parser)
11
12 add_executable(blacklist_compiler
13   ${BLACKLIST_COMPILER_SRC}
14   ${BISON_blacklist_parser_OUTPUTS}
15   ${FLEX_blacklist_lexer_OUTPUTS})
16
17 set(BLACKLIST_FILE ${...}/mf_blacklist.b)
18 set(BLACKLIST_OBJ ${...}/C_code.o)
```

```
19 add_custom_command(
20   OUTPUT ${BLACKLIST_OBJ}
21   COMMAND ${CMAKE_COMMAND}
22     -E copy ${BLACKLIST_FILE}
23     ${...}/mf_blacklist.b
24   COMMAND ${UTIL_DIR}/makeup/blcomp
25     -c -C ${CMAKE_C_COMPILER}
26     -x $<TARGET_FILE:blacklist_compiler>
27     mf_blacklist.b
28   DEPENDS
29     blacklist_compiler
30     ${BLACKLIST_FILE}
31   WORKING_DIRECTORY
32     ${...}
33   VERBATIM)
```

Code generation: ODB

- Generator tool `odb98.x` is built in a regular way
- C code is generated in three steps: tables, views and `<...>_Sstatic.c`
- List of SQL tables should be known at configure time
- It could be obtained from `*.h` files within the `ddl.<...>` directories

```
set(cma_limits
  NMXENKF NMXENDA NMXFCDIAG NMXUPD)
list(JOIN cma_limits "|" cma_limits_regex)
file(STRINGS ddl.${ARG_ODBASE}/cma.h limits
  REGEX "^SET[ \t]+\\$(${cma_limits_regex})[ \t]*=[ \t]*[0-9]")
foreach(item IN LISTS limits)
  string(REGEX MATCH
    "^SET[ \t]+\\$([A-Za-z0-9_]+)[ \t]*=[ \t]*([0-9]+)"
    match_limit "${item}")
  if(match_limit)
    set(${CMAKE_MATCH_1} ${CMAKE_MATCH_2})
  else()
    message(FATAL_ERROR "Unable to parse limit: ${item}")
  endif()
endforeach()
```

Code generation: ODB

```
16  file(GLOB cma_headers LIST_DIRECTORIES false ddl.${ARG_ODBASE}/*.h)
17  foreach(header IN LISTS cma_headers)
18    file(STRINGS ${header} table_defs REGEX "^[ \t]*CREATE[ \t]+TABLE")
19    string(REGEX REPLACE "\\[" "@@@LBRA@@@" table_defs "${table_defs}")
20    string(REGEX REPLACE "\\]" "@@@RBRA@@@" table_defs "${table_defs}")
21    foreach(definition IN LISTS table_defs)
22      string(REGEX MATCH
23        "^[ \t]*CREATE[ \t]+TABLE[ \t]+([A-Za-z0-9_]+)"
24        match_name "${definition}")
25      if(match_name)
26        set(table_name ${CMAKE_MATCH_1})
27        string(REGEX MATCH
28          "@@@LBRA@@@([0-9]+):\\$([A-Za-z0-9_]+)@@@RBRA@@@"
29          match_vector "${definition}")
30        if(match_vector)
31          set(first_member ${CMAKE_MATCH_1})
32          set(last_member ${${CMAKE_MATCH_2}})
33          foreach(member RANGE ${first_member} ${last_member})
34            list(APPEND expected_tables
35              "${ARG_ODBASE}_T_${table_name}_${member}.c")
36          endforeach()
37        else()
38          list(APPEND expected_tables "${ARG_ODBASE}_T_${table_name}.c")
39        endif()
40      endif()
41    endforeach(definition)
42  endforeach(header)
```

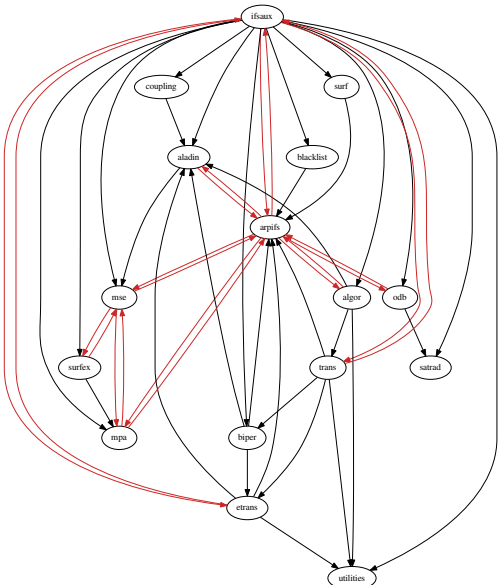
Code generation: ODB

```
43  set(cma_generated
44    ${expected_tables}
45    ${ARG_ODBASE}.c
46    ${ARG_ODBASE}.h)
47
48  set(cma_source_dir ${CMAKE_CURRENT_SOURCE_DIR}/ddl.${ARG_ODBASE})
49  set(cma_generated_dir ${CMAKE_CURRENT_BINARY_DIR}/ddl.${ARG_ODBASE})
50  file(MAKE_DIRECTORY ${cma_generated_dir})
51
52  set(cma_generated_src "")
53  foreach(source_file IN LISTS cma_generated)
54    list(APPEND cma_generated_src "${cma_generated_dir}/${source_file}")
55  endforeach()
56
57  set(cma_ddl_file ${cma_source_dir}/${ARG_ODBASE}.ddl)
58  set(odb_compiler_flags_file ${cma_source_dir}/odb98.flags)
59  set(odb_compiler_flags ODB_COMPILER_FLAGS=${odb_compiler_flags_file})
60  add_custom_command(
61    OUTPUT ${cma_generated_src} ${cma_generated_dir}/${ARG_ODBASE}.ddl_
62    COMMAND ${odb_compiler_flags} $<TARGET_FILE:odb_compiler> -O3 -C
63      -DCANARI -UECMWF -c -I ${cma_source_dir} -l ${ARG_ODBASE}
64      -o ${cma_generated_dir} ${cma_ddl_file}
65    DEPENDS odb_compiler ${cma_ddl_file} ${odb_compiler_flags_file}
66    WORKING_DIRECTORY ${cma_generated_dir}
67    COMMENT "Generating ${ARG_ODBASE} tables C-sources"
68    VERBATIM)
```

Code generation: ODB

```
69  file(GLOB cma_sqls LIST_DIRECTORIES false ddl.${ARG_ODBASE}/*.sql)
70  set(cma_static_generated_src ${cma_generated_dir}/${ARG_ODBASE}_Sstatic.c)
71  add_custom_command(
72    OUTPUT ${cma_static_generated_src}
73    COMMAND ${UTIL_DIR}/makeup/gen_static ${ARG_ODBASE} ${cma_sqls}
74    DEPENDS ${UTIL_DIR}/makeup/gen_static ${cma_sqls}
75    WORKING_DIRECTORY ${cma_generated_dir}
76    VERBATIM)
77  foreach(sql_file_path IN LISTS cma_sqls)
78    get_filename_component(file_name_we  ${sql_file_path} NAME_WE)
79    get_filename_component(sql_file_name ${sql_file_path} NAME)
80    set(compiled_sql_file
81      "${cma_generated_dir}/${ARG_ODBASE}_${file_name_we}.c")
82    add_custom_command(
83      OUTPUT ${compiled_sql_file}
84      COMMAND ${CMAKE_COMMAND} -E copy_if_different
85        ${sql_file_path} ${cma_generated_dir}
86      COMMAND ${odb_compiler_flags} $<TARGET_FILE:odb_compiler> -O3 -C
87        -DCANARI -UECMWF -c -I ${cma_generated_dir} -l ${ARG_ODBASE}
88        -o ${cma_generated_dir} -i -w ${sql_file_name}
89      DEPENDS odb_compiler ${sql_file_path}
90        ${ODB_COMPILER_FLAGS_FILE} ${cma_generated_dir}/${ARG_ODBASE}.ddl_
91      WORKING_DIRECTORY ${cma_generated_dir}
92      VERBATIM)
93    list(APPEND cma_compiled_sql_to_c_src ${compiled_sql_file})
94  endforeach()
```

High-level structure of the source code



Main dependencies between the code components

- a lot of cycles between the various libraries
- Makeup build operates on the source-file level
- CMake resolves inter-project dependencies over targets
- Direct translation to CMake is not possible thanks to Fortran modules

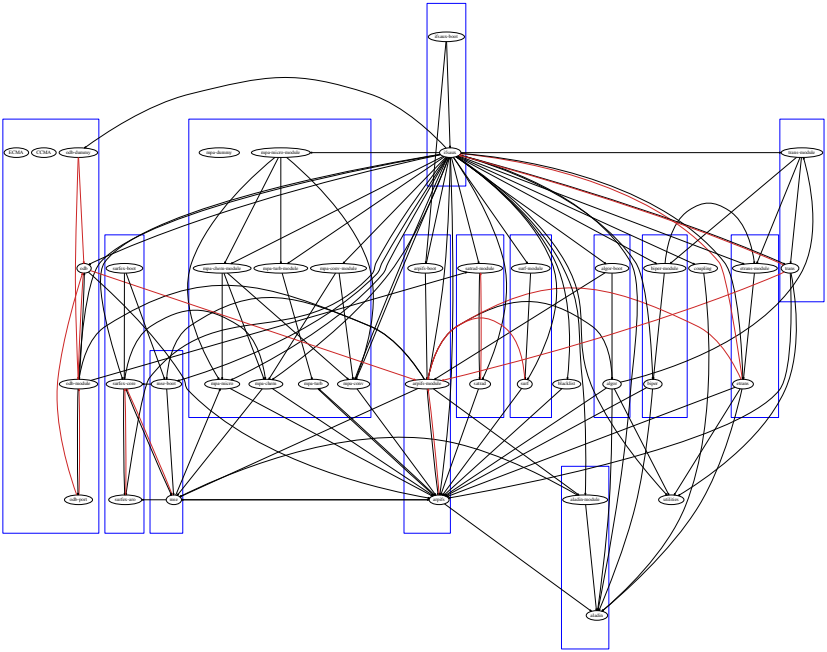
Why it is difficult to deal with cycles and modules

```
1  add_library(A STATIC ${LIB_A_SRC})
2  add_library(B STATIC ${LIB_B_SRC})
3  target_link_libraries(A B)
4  target_link_libraries(B A)
5
6  add_executable(main ${PROG_SRC})
7  target_link_libraries(main A)
```

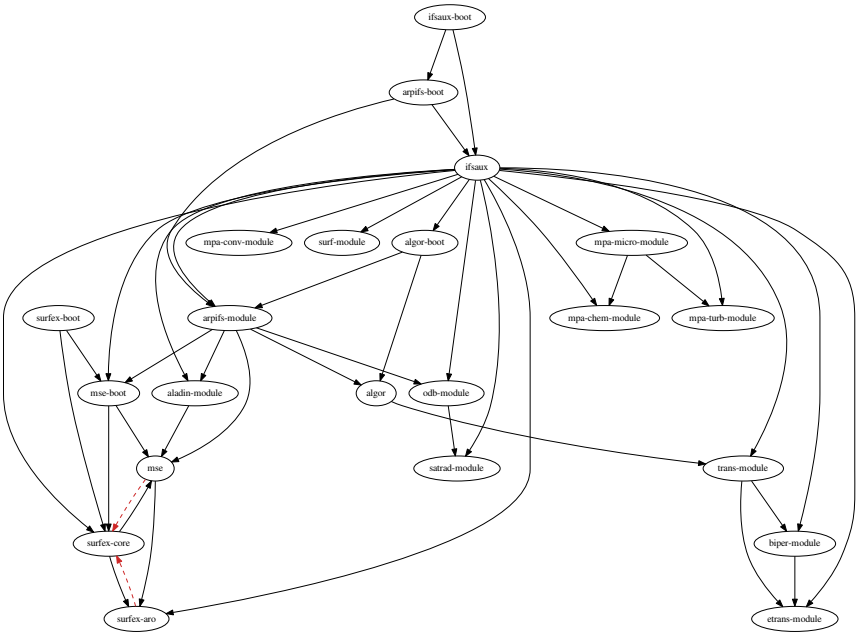
- CMake allows cyclic dependencies for static libraries and resolves repetitions at the link time: `main` is linked to `A B A B`
- Compilation order is not enforced

- If a component depends on a Fortran module from another component this dependency enforces the compilation order
- CMake is unable to compile the component in the correct order because of the cyclic dependency
- Build system behaviour becomes indeterministic, it could succeed or fail depending on the number of build processes

Split components to avoid cycles between Fortran modules



Now module libraries form a DAG



Note on SURFEX and MSE

- SURFEX and MSE libraries form a strongly coupled pair
- MSE is naturally a library that depends on SURFEX
- But SURFEX calls routines from MSE for IO-related tasks
- It is not possible to resolve cycles without link-time tricks

Note on SURFEX and MSE

- SURFEX and MSE libraries form a strongly coupled pair
- MSE is naturally a library that depends on SURFEX
- But SURFEX calls routines from MSE for IO-related tasks
- It is not possible to resolve cycles without link-time tricks

- Untying could be done by introducing polymorphic IO in SURFEX
- Would require substantial code refactoring
- But modified code has potential for a more clear structure

Note on SURFEX and MSE

Current IO interface

```
! write_surf.F90
INTERFACE WRITE_SURF
  ! Modified for clarity
  MODULE PROCEDURE WRITE_SURFX1
END INTERFACE WRITE_SURF

SUBROUTINE WRITE_SURFX1(HPROGRAM, <...>)
! ...
  IF (HPROGRAM=='AROME ') THEN
#ifdef SFX_ARO
    CALL WRITE_SURFX1_ARO(<...>)
#endif
  END IF
  IF (HPROGRAM=='OFFLIN') THEN
#ifdef SFX_OL
    CALL WRITE_SURFX1N1_OL(<...>)
#endif
  END IF
! ...
END SUBROUTINE WRITE_SURFX1

! Usage:
CALL WRITE_SURF(HPROGRAM, <...>)
```

Polymorphic IO interface

```
TYPE, ABSTRACT :: SURFEX_IO_INTERFACE_t
  CONTAINS
    PROCEDURE(IWRITE), PASS, DEFERRED :: &
      WRITE_SURFX1
    ! ...
    GENERIC :: WRITE_SURF => &
      WRITE_SURFX1, <...>
END TYPE SURFEX_IO_INTERFACE_t

ABSTRACT INTERFACE
  SUBROUTINE IWRITE(THIS, <...>)
    IMPORT :: SURFEX_IO_INTERFACE_t
    CLASS(SURFEX_IO_INTERFACE_t) :: THIS
    ! ...
  END SUBROUTINE IWRITE
END INTERFACE

! Usage
CLASS(SURFEX_IO_INTERFACE_t), &
  POINTER :: IO => NULL()
! Allocate once, during init
ALLOCATE(AROME_IO_t :: IO)

CALL IO%WRITE_SURF(<...>)
```

POC CMake build for ALADIN-HIRLAM NWP system

- Only MASTERODB is built for this test case
- Two-step linking of the Makeup build with automatic generation of dummy routines is replaced by pre-generated `dummies.c`

```
> cd Harmonie-cmake
> mkdir build && cd build
> FC=ifort CC=icc cmake ../src/ -DCMAKE_BUILD_TYPE=Release -DNETCDF_DIR=<...> \
  -DHDF5_DIR=<...> -Deccodes_DIR=<...> -Dbufr_DIR=<...> -Dgribex_DIR=<...>
> # ... CMake output is omitted ...
> make -j32
```

POC CMake build for ALADIN-HIRLAM NWP system

- Only MASTERODB is built for this test case
- Two-step linking of the Makeup build with automatic generation of dummy routines is replaced by pre-generated `dummies.c`

```
> cd Harmonie-cmake
> mkdir build && cd build
> FC=ifort CC=icc cmake ../src/ -DCMAKE_BUILD_TYPE=Release -DNETCDF_DIR=<...> \
  -DHDF5_DIR=<...> -Deccodes_DIR=<...> -Dbufr_DIR=<...> -Dgribex_DIR=<...>
> # ... CMake output is omitted ...
> make -j32


> # ... output is omitted ...
> [100%] Built target odb-port-static
> Scanning dependencies of target master
> [100%] Building Fortran object arpifs/CMakeFiles/master.dir/programs/master.F90.o
> [100%] Linking Fortran executable master
> [100%] Built target master
>
```

Timings for CMake and Makeup builds

Tests were performed on nebula – MET's research HPC

	Makeup	CMake	
		make	Ninja
Configuration	00:10:00	00:01:30	00:01:40
Full build	01:15:00	00:20:00	00:18:27
No-op build	00:06:50	00:00:40	00:01:35
Incremental build	00:07:00	00:01:20	00:02:22

Note that Makeup builds all executables and CMake only MASTERODB
(but the full set of libraries is built in both cases)

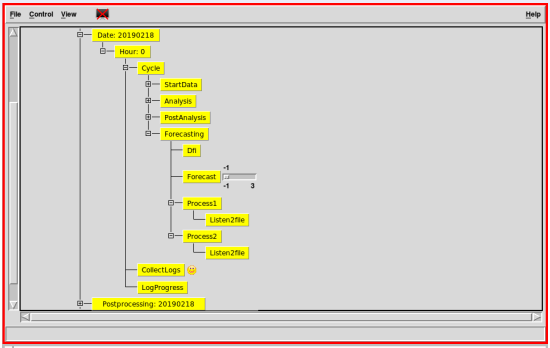
If you got interested in CMake for Fortran projects and want to try it but have a feeling that building the whole ALADIN-HIRLAM system is a bit too much...

...you could check the CMake-powered fork of the Open-SURFEX platform

available under

https://github.com/joewkr/open-SURFEX

Questions?




```
[ 41%] Built target coupling-static
[ 41%] Built target mpa-conv-module-static
[ 42%] Built target mpa-micro-module-static
[ 42%] Built target mpa-turb-module-static
[ 43%] Built target surf-module-static
[ 43%] Built target surf-static
[ 44%] Built target mpa-chem-module-static
[ 44%] Built target mpa-conv-static
[ 47%] Built target arpifs-module-static
[ 47%] Built target aladin-module-static
[ 47%] Built target mpa-turb-static
[ 47%] Built target algor-static
[ 48%] Built target mse-boot
[ 48%] Built target odb-module-static
[ 48%] Built target odb-static
[ 49%] Built target trans-module-static
[ 49%] Built target trans-static
[ 49%] Built target biper-module-static
[ 49%] Built target biper-static
[ 50%] Built target satrad-module-static
[ 50%] Built target etrans-module-static
[ 50%] Built target etrans-static
[ 52%] Built target satrad-static
[ 53%] Built target utilities-static
[ 78%] Built target surfex-core-static
[ 79%] Built target mpa-chem-static
[ 97%] Built target arpifs-static
[ 98%] Built target mse-static
[ 98%] Built target mpa-micro-static
[ 98%] Built target surfex-static
[ 99%] Built target aladin-static
[100%] Built target odb-port-static
[100%] Built target master
[sm_yurba@nebula build]$
```