Extracting a test case from a subroutine in IAL

Cossevin Erwan, Penigaud Nicolas

Why extracting a test case ?

- Extract a small test case from the code => run a subroutine outside of the whole code
- When you have a tricky bug, it can be useful to extract just one routine from the code and run the code just for this routine
- It’s easier to work on a routine rather than all the code
- When you want to port some code on GPU, you can start with a small test case

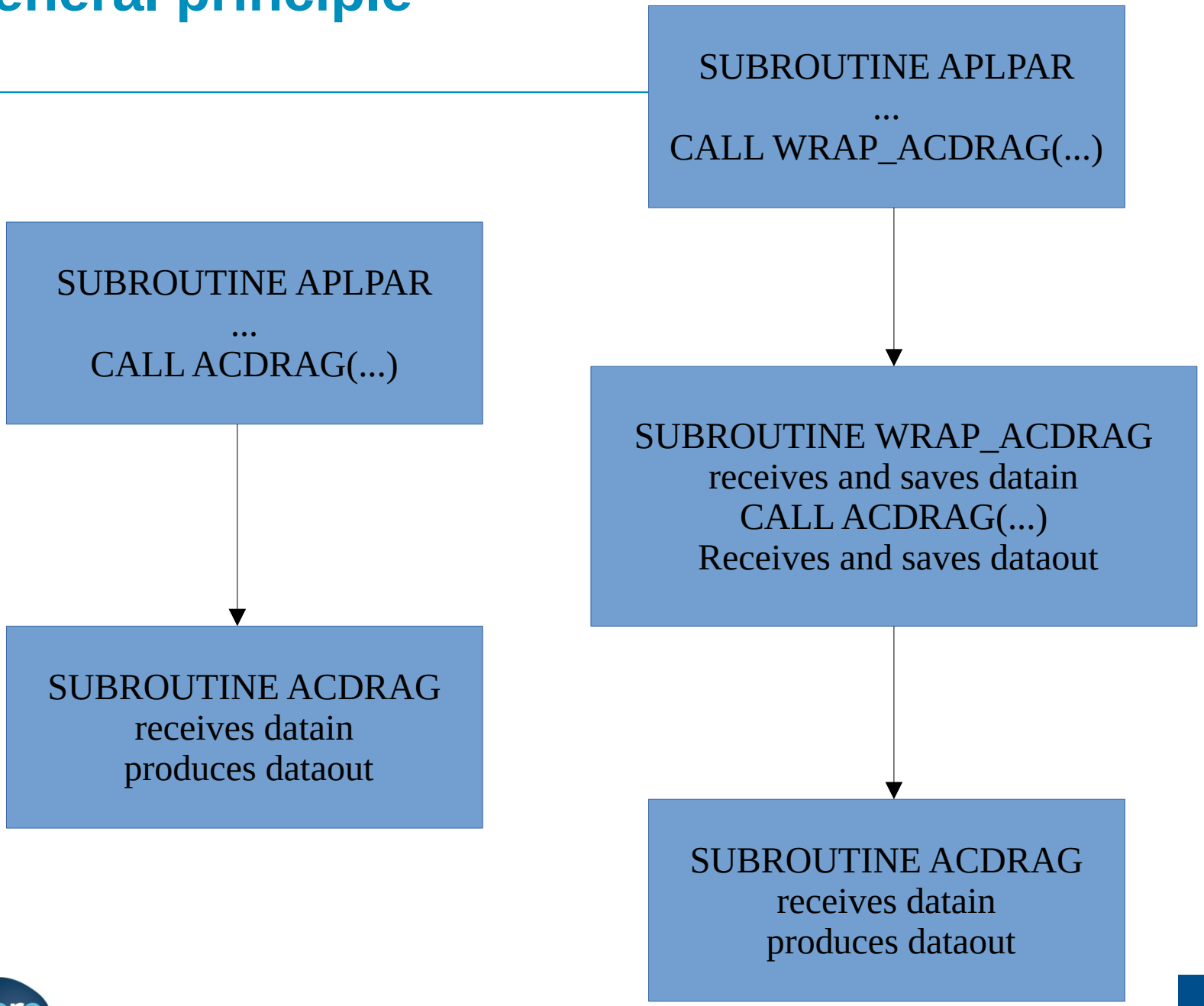
General principle

Two steps :

A) extract the data : Replace the call to the subroutine by a call to a wrapper that saves the data and then call the subroutine

B) Create the test case : create a new repository with the code from the model

General principle



Example

The extracted routine can be converted to GPU code with fxtran-acdc. In this practical, we will use the routine generated by fxtran-acdc, and other possibilities.

One such possibility is to check results between a CPU run and a GPU run.

By default, results are not bit-reproducible between CPU and GPU. Identified sources of non-reproducibility :
- matrix multiplies using BLAS or CUBLAS
- special functions (log, sin, cos, exp…)
- order of sums with the NVIDIA compiler.

The matrix multiplies can be replaced by their non-optimized version (3 loops sums and multiplies), which is bit-reproducible between CPU and GPU

Example

The script can replace the special functions by a bit-reproducible version (using only the 4 usual operations and the square root, which are bit-reproducible in IEEE 754).

For example, COS will be replaced by FXTRAN_ACDC_BR_COS, which is defined in a Fxtran-acdc library.

Order of operations, even in -O0 :
- NVDIA compiler : a+b+c+d computed as (a+b)+(c+d) for performance reasons ;
- other compilers : a+b+c+d computed as (((a+b)+c)+d)

This can lead to roundup differences.

The script can remove this source of non-reproducibility by adding parentheses : a+b+c+d systematically rewritten as (((a+b)+c)+d).

Questions ?

Appendix : Solution code

- The branch in which the wrapper is coded is : https://github.com/ecossevin/IAL/tree/create_wrapper_acdrag

- The branch with the extracted test case is :

  https://github.com/ecossevin/acdrag/tree/master

- Some files to help (starting point for each step mentionned):

  to extract the data : https://github.com/ecossevin/acdrag/tree/extract_test_case

  to generate util methods:

  https://github.com/ecossevin/acdrag/tree/generate_util

  to run the test_case: https://github.com/ecossevin/acdrag/tree/run_test_case

- You can find all that at : /perm/rma1/test_case_working_week (extraction of acdrag miniapp) on the ecmwf cluster and /home/soa1/practical_working_week (use of acdrag miniapp)