Swiss Confederation
Federal Department of Home Affairs
Federal Office of Meteorology and Climatology MeteoSwiss

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Reproducible climate and weather simulations: an application to the COSMO model

C. Charpilloz*, A. Arteaga*,†, O. Fuhrer*, T. Montek‡, and C. Harrop‡

* Federal Office of Meteotology and Climatology MeteoSwiss, † Swiss Federal Institute of Technology Zurich,
‡ Cooperative Institute for Environmental Sciences

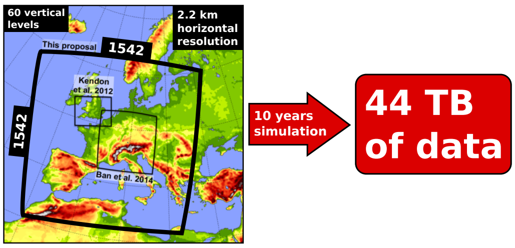
Fonds national suisse
Schweizerischer Nationalfonds
Fondo nazionale svizzero
Swiss National Science Foundation

CIRES

Introduction

Environment with changing climate requires the analysis of convection-resolving simulations at continental scale. However the **huge amount of produced data** makes the analysis of the simulation impractical as **it cannot be stored**.

We propose a new framework where data are transparently accessed or re-simulated with reruns on demand: **storage is traded for computational effort**.

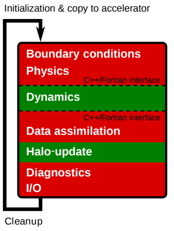


10 years simulation → 44 TB of data

Problem: convection-resolving simulations → large computational costs → restriction to small domains or short time scales.
Solution: adapt the COSMO model to use the largest available supercomputers systems such as hybrid CPU-GPU architectures [2].

Material: the COSMO model

COSMO is a non-hydrostatic atmospheric model based on finite difference solvers and stencils computations.



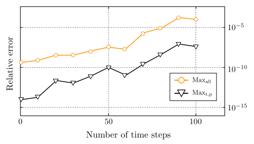
Technologies

C++	Fortran	Arch.
NVCC (CUDA)	Cray, PGI (OpenACC)	GPU
GCC	GCC, Cray, PGI	CPU

Problems:
(i) data access may trigger reruns of the simulation on different architectures.

(ii) different architectures → different compilers → different computations

(iii) climate simulations are highly non-linear → small initial differences in re-runs will rapidly grow into larger differences



Solution: adapt COSMO to produce **bit reproducible results** on these architectures [1].

Source of non-reproducibility

COSMO is a good candidate model for reproducibility: no concurrent access to variables between threads and no reduction operations.

But sources of non-reproducibility exist in COSMO:

(i) the transcendental functions need to be replaced by a portable set of functions that should be used regardless the compiler or architecture (C++ & Fortran).

(ii) the problem of intrinsic operators must be replaced by a function call then the problem is related to (i) (Fortran).

(iii) compiler optimizations alternate code depending on data size, strength reduction, IEEE compliance, usage of floating point contractions. They should be controlled with flags and/or directives (C++ & Fortran).

(iv) the reassociation of operation has to be forbidden (Fortran):

$$x \circledast y \circledast z \neq y \circledast z \circledast x \Leftrightarrow r(r(x \star y) \star z) \neq r(x \star r(y \star z))$$

Example of reassociation applied by the compiler:

(a)
```
zqst = siau + sdau + sagg - ssmelt + sicri
       + srcri + srim + ssdep + srfrz
```

(b)
```
vmovsd    -560(%rbp), %xmm0
vaddsd    -552(%rbp), %xmm0, %xmm0
vaddsd    -544(%rbp), %xmm0, %xmm0
vsubsd    -536(%rbp), %xmm0, %xmm0
vaddsd    -528(%rbp), %xmm0, %xmm0
vmovsd    -520(%rbp), %xmm1
vaddsd    -512(%rbp), %xmm1, %xmm1
vaddsd    -504(%rbp), %xmm1, %xmm1
vaddsd    -496(%rbp), %xmm1, %xmm1
vaddsd    %xmm0, %xmm1, %xmm0
vmovsd    %xmm0, -488(%rbp)
```

(c)
```
zqst = (siau + sdau + sagg - ssmelt + sicri) +
       (srcri + srim + ssdep + srfrz)
```

The assembly (b) generated from the expression (a) corresponds to the expression (c):

Method: preprocessing the code and optimization flags

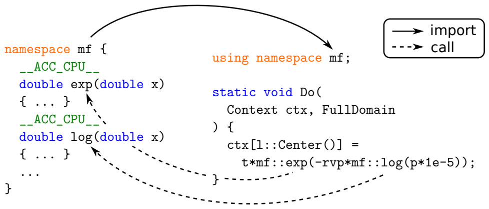
The dynamics (C++/CUDA)

C++ standard is restrictive regarding the code optimizations. Two step are needed:

(i) deactivating the FMA:

with GNU: -fp-contract=off and with NVCC: -nofma

(ii) providing portable transcendental functions:



```
namespace mf {
  __ACC_CPU__
  double exp(double x)
  { ... }
  __ACC_CPU__
  double log(double x)
  { ... }
  ...
}
```

```
using namespace mf;

static void Do(
  Context ctx, FullDomain
) {
  ctx[l::Center()] =
    t*mf::exp(-rvp*mf::log(p*1e-5));
}
```

The physics (Fortran/OpenAcc)

Fortran compilers have much more freedom in reorganizing the expressions or instructions in order to optimize the execution speed. Hence the code must be preprocessed.

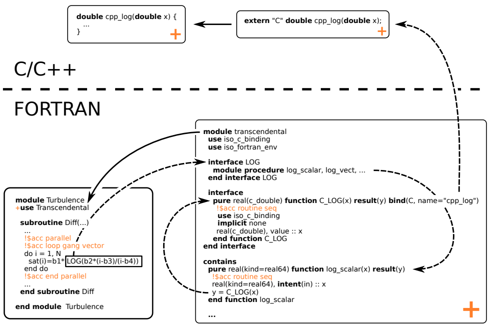
(i) The order of evaluation must be unique to ensure the generation of a unique AST. We add parenthesis:

```
ztu8 = pa2c * pcb1 + ztd6 * ztu2 + ztd7 * ztu4
```

↓

```
ztu8 = ((((pa2c * pcb1) + (ztd6 * ztu2)) + (ztd7 * ztu4)))
```

(ii) The transcendental functions' calls must be shadowed by a portable implementation:



```
double cpp_log(double x) {
  ...
}
```

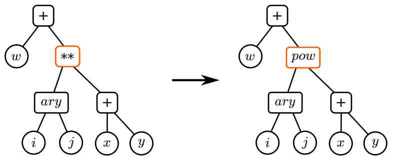
```
extern "C" double cpp_log(double x);
```

```
module Turbulence
+use Transcendental

 subroutine Diff(...)
  ...
  !$acc parallel
  !$acc loop gang vector
  do i = 1, N
   sat(i)=b1*LOG(b2*(i-b3)/(i-b4))
  end do
  !$acc end parallel
  ...
 end subroutine Diff

end module  Turbulence
```

```
module transcendental
  use iso_c_binding
  use iso_fortran_env

  interface LOG
    module procedure log_scalar, log_vect, ...
  end interface LOG

  interface
    pure real(c_double) function C_LOG(x) result(y) bind(C, name="cpp_log")
     !$acc routine seq
     use iso_c_binding
     implicit none
     real(c_double), value :: x
    end function C_LOG
  end interface

  contains
  pure real(kind=real64) function log_scalar(x) result(y)
   !$acc routine seq
   real(kind=real64), intent(in) :: x
   y = C_LOG(x)
  end function log_scalar

  ...
```

(iii) As one cannot shadow intrinsic operators in Fortran we need to replace them. In COSMO it's the exponentiation operator:

$$w{+}ary(i,j){*}{*}(x{+}\ y) \longrightarrow w{+}pow(ary(i,j),(x{+}\ y))$$



Results

Full bit-reproducibility has been achieved for COSMO in some setups. The following matrix shows which setups are reproducible:

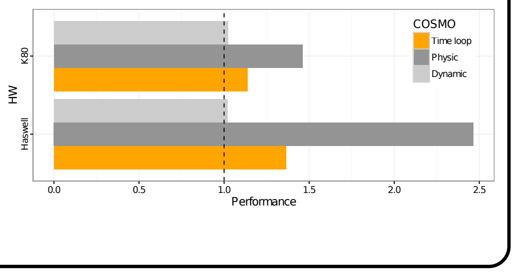
Tab. 1 - Reproducibility matrix across compilers and architecture. Each color represent a match with another setup. The test machine has Intel Haswell E5-2690v3 CPU and Nvidia Tesla K80 GPU.

	GNU	PGI	Cray	NVCC
GPU	☒		🟧	◼
CPU	🟦 ◼	🟦	🟦 🟧	☒

Tab. 2 - Flags used to obtain the reproducible setups represented by the colors. The reproducibility is achieved on a preprocessed code (see Method).

Setup	GNU	PGI	Cray	NVCC
🟦	march=native, ffp-contract=off, O0	Kieee, concur=noaltcode,noassoc, nofma, O0, nolre	nopattern, fp0, O0, noaggress, flex_mp=intolerant	
◼	march=native, ffp-contract=off, O0			fmad=false, ftz=false, O0, prec-div=true, prec-sqrt=true
🟧			fp0, O2, flex_mp=intolerant	

Penalty of the bit-reproducibility setup when compared to the "native" version of the setup with the Cray compiler across CPU and GPU (1.0 indicates similar runtime):



References

[1] Arteaga, A., Fuhrer, O., and Hoefler, T.: Designing bit-reproducible portable high-performance applications, in: Parallel and Distributed Processing Symposium, 2014 IEEE 28th International, pp. 1235-1244, IEEE, 2014.

[2] Lapillonne, X., Fuhrer, O.: Using compiler directives to port large scientific applications to GPUs: An example from atmospheric science, in: Parallel Processing Letters, 24, 2014.

[3] Li, R., Liu, L., Yang, G., Zhang, C., Wang, B.: Bitwise identical compiling setup: prospective for reproducibility and reliability of Earth system modeling, in: Geoscientific Model Development, 9, pp. 731-748, 2016.

Platform for Advanced Scientific Computing 2017, Lugano, Switzerland