COSMO in Single Precision

Stefan Rüdisühli, André Walser, and Oliver Fuhrer

MeteoSwiss, Zürich

Abstract

Reducing arithmetic precision in a numerical weather prediction model, and thus reducing the number of bytes required to store a floating point number, can be advantageous for an application such as COSMO both in terms of runtime and memory consumption. But, since the COSMO model has been written and applied only using double precision floating point numbers, reducing arithmetic precision requires careful consideration. In this article, we present the modifications necessary to the COSMO model to reduce the arithmetic precision of floating point numbers to single precision. Using concrete examples from the current code that did not work in single precision, we illustrate critical algorithmic patterns that should receive special attention from developers in order not to rely on double precision in the future. We present results from detailed tests as well as a standard verification of the newly developed model version. Results indicate that the new version exhibits the same forecast skill as the reference version, both in single and double precision mode. In single precision, the runtime drops to $\approx 60\%$ and memory consumption is reduced considerably, as compared to the double precision mode.

1. Introduction

Numerical Weather Prediction (NWP) models consume immense amounts of computer resources and, as a consequence, electrical energy. Model development and application is thus constrained by both monetary and technical limits. Considering the ever-increasing demand for computer resources, fueled by current trends such as cloud-resolving modeling or ensemble predictions, techniques to make models faster and more energy efficient are highly welcome.

One approach which promises a significant speedup is running models with reduced arithmetic precision. Current computer hardware typically supports floating point computations in single precision (SP) and double precision (DP). While it is still customary to use DP for NWP, reducing arithmetic precision to SP has several advantages. First and foremost, less information has to be moved to the floating point unit of the microprocessor in order to perform a computation. Second, microprocessors are typically capable of performing more floating point operations per second (FLOPS) in reduced precision. Third, the memory consumption of an application can be significantly reduced. Often, reducing the arithmetic precision of an application can be achieved with relatively little changes to the code, as compared to other approaches such as code optimization or porting to more efficient hardware such as GPUs. The latter often requires substantial code modifications or even partial rewrites. Several other weather and climate models are already capable of running in SP [5, 3] or are in the process of being adapted to SP [2].

In this article, we present the steps which are required to adapt the COSMO model to run in SP. In the new code version, the working precision (WP) of the model can be chosen using a single switch. By means of extensive validation, we show that our adapted code in DP can replace the previous code, and that it's skill in SP should be sufficient for many applications. We confirm and build upon findings of preliminary work, which showed that not only the dynamical core [4], but the whole model can be run in SP (Despraz and Fuhrer, *pers. comm.*) without significant loss in forecast quality, and with only few changes to the code.

2. Background

2.1. Floating Point Numbers and Precision

Floating point numbers (FPNs) on computers are stored in binary form. They consist of three parts: the sign (plus/minus), the exponent (order of magnitude), and the mantissa (significant digits). On most common hardware architectures, the representation of FPNs follows a standard [1] on most current microprocessors. This standard defines how FPNs are stored in binary form and how operations between two FPNs have to

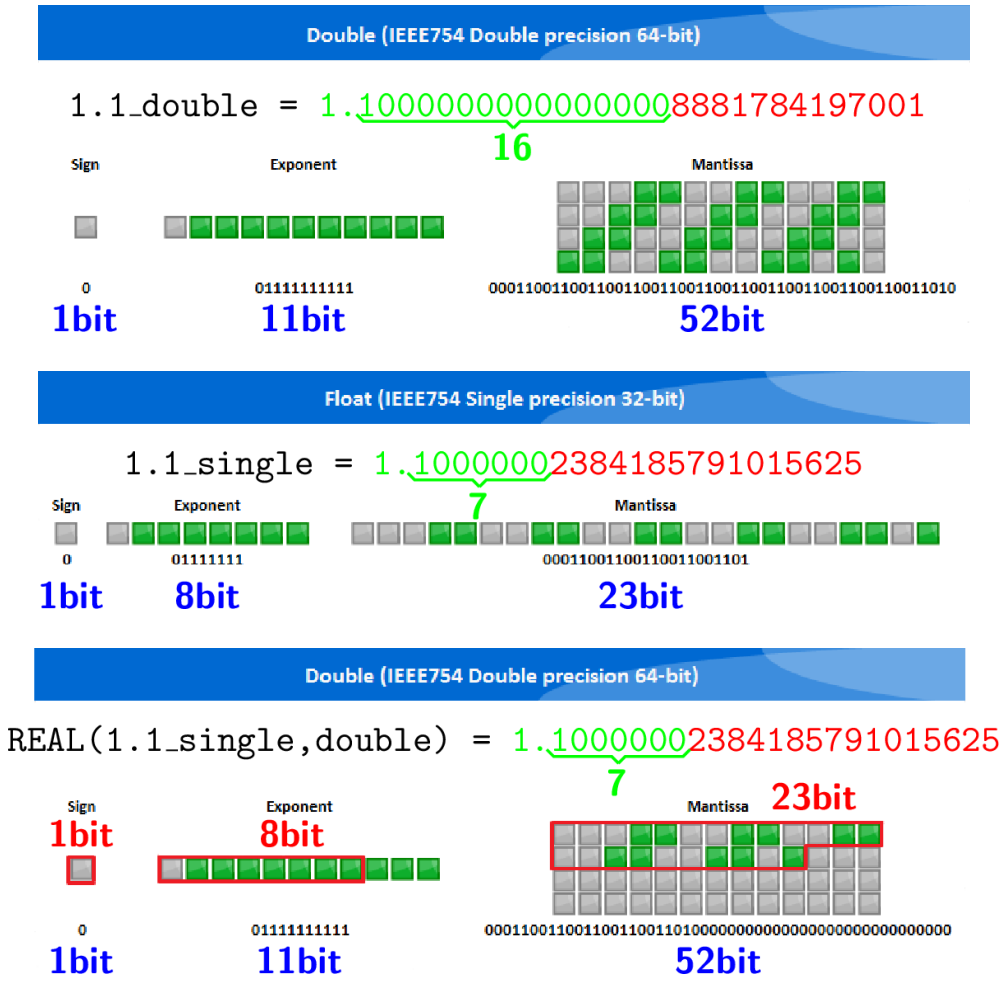

FIGURE 1: Binary representation of DP and SP floats on computers using the example of 1.1. Each square corresponds to one bit, grey standing for 0 and green for 1. The number of bits of all three components of FPNs is indicated. They add up to 64 and 32 bits for DP and SP, respectively. In DP (top), 1.1 is accurate to 16 decimal digits behind the decimal point, but only to 7 decimal digits in SP (center). If 1.1 in SP is assigned to a DP variable (bottom), only the first 23 bits of the mantissa carry information (red border). All bits outside this area are 0, which reduces the accuracy to 7 decimal digits behind the decimal point. *Source:* `http://www.binaryconvert.com`

be implemented. The magnitude and precision ranges of FPNs in DP and SP are listed in Table 1. How FPNs are stored in binary form is illustrated in Figure 1, using the number 1.1 as an example. The top and center panels show how the example FPN is stored in binary form in DP and SP, respectively. If converted from binary to decimal form, the number of significant decimal digits after the floating point is 16 and 7, respectively, determined by the size in bits of the mantissa. The digits beyond (marked red) may seem to be random, but they are not. They are the deterministic product of the conversion from binary to decimal. This is illustrated in the bottom panel, where 1.1 in SP resolution is shown in DP. The difference between the FPNs resulting from the conversion (i.e. `1.1_double-REAL(1.1_single,double)`) stems from the bits which are 1 (green) in the top, but 0 (grey) in the bottom panel.

precision	total size	max	min	digits	precision
single	32 bit	10^{38}	10^{-38}	7.2	10^{-7}
double	64 bit	10^{308}	10^{-308}	16.0	10^{-16}

TABLE 1: Magnitude and precision ranges of SP and DP FPNs according to the IEEE 754 standard. The maximum and minimum magnitudes are determined by the size of the exponent, the number of digits by the mantissa. The precision of FPNs on the order of 1.0 is determined by the number of digits.

2.2. Precision in Fortran

In Fortran, FPNs are represented by the basic type real. Typically reals are SP by default, although this may depend on the specific compiler used. To ensure a program runs in a certain precision, the kind of every real variable as well as of every FPN in the program has to be declared explicitly, which determines the number of bytes used to store a real. It is set with an integer parameter. Often, it directly corresponds to the size a FPN in bytes, i.e. 4 and 8 for SP and DP, respectively. As this is platform-dependent, however, intrinsic routines are provided to obtain the correct values. The kind parameter used in COSMO is called `ireals`[3]. How it is used to set the precision of variables, of FPNs, as well as in type conversions is illustrated with the following line of code. A real variable is defined and initialized to the sum of a FPN and an integer variable.

```
REAL(KIND=ireals) :: var = 3.5_ireals + REAL(intvar,ireals)
```

If the '`_ireals`' following a FPN (3.5 in this case) is omitted, it is typically defined in SP by default. This introduces arbitrary inaccuracies of relative order of magnitude $O\left(10^{-7}\right)$ as illustrated in Figure 1.

2.3. Epsilons

Epsilons are small numbers used for various numerical purposes in a code. They are used to account for inaccuracies resulting from the limited ability of computers to represent FPNs. Furthermore, they are used as tolerances in both numerical and physical contexts. Three examples of popular applications of epsilons are the following.

1. In divisions to avoid division by zero (DBZ), e.g. `x = y / MAX(z,eps)`[4].
2. In comparisons of FPNs, e.g. `IF (ABS(a-b) < eps) equal = .TRUE.`).
3. In iterations as abort criteria, e.g.
   `IF (MAXVAL(ABS(p_new(:,:,:)-p_old(:,:,:))) < eps_abort) EXIT.`

Two basic kinds of epsilons can be distinguished. On the one hand, there are those epsilons the magnitude of which is only determined by the precision. On the other hand, many epsilons additionally have some meaning in an algorithm or physical context, which puts additional constraints on the magnitude. These two basic kinds will henceforth be referred to as *precision-limited* and *algorithmic* epsilons. Note that algorithmic epsilons are precision-limited, too, as, after all, all FPNs are. This means that the subsequently described limitations apply to them as well, but *precision-limited* will refer exclusively to those epsilons upon which no further limitations of algorithmic or physical nature are imposed.

Precision-limited epsilons can be further classified according to the factor which determines their minimal magnitude. To which group a specific epsilon belongs depends on it's context of use. Either the limiting factor is the minimal order of magnitude (*range-limited*) or the maximal number of decimal digits (*digit-limited*) that can be resolved, i.e. the exponent or mantissa, respectively. Range-limited epsilons are used to avoid floating point exceptions (FPEs) in operations where zero is not allowed as an operand, such as divisions or logarithms. Digit-limited epsilons, on the other hand, are used to account for inaccuracies in comparisons of FPNs.

The epsilon in the first of the three examples above is range-limited, and that in the second is digit-limited. Both are precision-limited epsilons. That in the third example is also digit-limited, but probably has some additional algorithmic constraints. However the distinction between precision-limited and algorithmic epsilons is not always straight-forward, and the context has to be taken into account in all but the most obvious cases. The epsilon in example two, for instance, might need to be bigger in certain algorithms with unusually large error growth, which would make it algorithmic. Also, using the minimal representable positive number larger than zero in the first example, may lead to erroneous results later on in the computation if the algorithm has not been designed carefully (for example by limiting the maximum result of the division to a physically reasonable value). However, it is most important to make this distinction in obvious cases where it is easy to draw the line, such as if the epsilon carries a physical unit.

Range-limited epsilons can theoretically be as small as 10^{-37} and 10^{-307} in SP and DP, respectively. However the magnitude of the dividend must be taken into account to avoid arithmetic overflow, e.g. if it is on the

[3] It is currently planned to unify the constant used to denote the WP across the COSMO, ICON, and 3DVAR codes. It is likely that in the future `ireals` will be renamed to `wp` which has the advantage of brevity.

[4] Note that this is only valid if `z` is strictly positive.

order of 10^7 the epsilon must be at least of magnitude 10^{-30} and 10^{-300}. As these values are still minuscule in a NWP model such as COSMO, it is sufficient to simply chose a value a couple of orders of magnitudes (e.g. 10^8) above the absolute minimum without further considerations, even in SP.

Digit-limited epsilons, on the other hand, must have a minimal magnitude – relative to 1.0 – of 10^{-7} and 10^{-16} in SP and DP, respectively. Taking into account the order of magnitude of the involved numbers is much more important than in the range-limited case, especially in SP, since 10^{-7} is not far from physical significance in many cases, for example for trace gas concentrations. Simply increasing the epsilon by some orders of magnitude to account for the magnitude of the other involved numbers is thus not a good solution. Rather the relative character of digit-limited epsilons should be accounted for in the way they are used in the formula, i.e. by not inserting them by addition (`a < b + eps`), but by multiplication (`a < (1.0 + eps)*b`). This much more robust implementation ensures that the epsilon does not vanish when large numbers are involved. Note that such a relative-epsilon implementation is more difficult when trying to determine the equality of two FPNs, e.g. `ABS(a-b) < eps`. Making the epsilon relative in this case would result for instance in `ABS(a-b) < ABS(a)*eps`, which is already quite complicated.

2.4. Numerical Errors

Numerical error growth can be minimized if certain formulas are written in a numerically robust form. What should be avoided wherever possible are subtractions of very similar numbers, as well as raising numbers to high powers. This is usually not a major concern in DP. In SP, however, error growth from such subtractions can be substantial, and high powers may even provoke arithmetic under- or overflow, leading to model crashes.

A good example to illustrate precision loss in subtractions of similar numbers is the computation of a small difference between two temperatures, e.g. `274.00000 - 273.15000 = 0.85000???`. In this case, three significant digits are immediately lost, as indicated by the three trailing digits in the result, which are subject to rounding error. An epsilon inserted by addition (which, unfortunately, is the usual practice) in such a formula in SP must be at least of absolute magnitude 10^{-5} in order not to completely vanish. This has to be considered when using epsilons in such cases by either increasing their magnitude or inserting them by multiplication. This is a common problem with temperature and pressure, which can easily be avoided if deviations from some reference are used instead of absolute values.

3. Changes to the Code

The COSMO model has been developed for and tested in DP. Therefore, there are many places in the code where something goes wrong in SP that works perfectly fine in DP. This has required us to conduct a variety of changes to the code to run COSMO in SP. These modifications can be broadly grouped into three categories.

- Obtain a pure DP code by adding all missing `ireals` declarations.
- Conduct various local changes.
  - Add epsilons to divisions to prevent FPEs by DBZ.
  - Adapt epsilons which might vanish in SP to be precision-dependent.
  - Optimize formulas that might cause numeric overflow or other problems.
- Implement mixed precision (MP) form of radiation.

3.1. Ireals declarations

The real kind parameter `ireals` in COSMO, which determines the WP, is set to DP by default. However, a large number of `ireals` declarations are missing in the current code, which introduces many SP reals into the model, thereby lowering the precision. The first step is therefore to add all these missing declarations in order to obtain a code running in pure DP. Because of the large number of missing declarations (see Table 2), we have written a script which automatically finds and declares all undeclared real variables and FPNs.

The consequence of the additional `ireals` declaration is that the model results of the new code differ from those of the old code (numerically, not meteorologically). This difference is due to the removal of the inaccuracies introduced by the undeclared reals, which corresponds to a perturbation of relative order of magnitude $\mathcal{O}\left(10^{-7}\right)$. Validation of this pure DP code by means of comparing results against simulations with random perturbations of similar magnitude is provided in Section 4.1.1. It is worth noting that these changes are the

only ones we have introduced which cause different model results in DP. All other code changes are neutral in DP and only have a significant effect in SP.

	nf	nf*	nold	del	del%	del%*
src_*	60	43	6276	4299	68.5	81.9
data_*	27	4	791	1141	144.3	967.0
rest	44	25	3026	682	22.5	26.6
total	133	72	10093	6122	60.7	76.2

nf	total number of files	del	number of _ireals added (delta)
nf*	number of changed files	del%	relative delta (all files)
nold	initial number of _ireals	del%*	relative delta (only changed files)

TABLE 2: Statistics of occurrences of `ireals` declarations before and after our modifications. The source files are sorted into three groups. Overall,we have added roughly 60% additional declarations.

3.2. Local Modifications

A number of local modifications are necessary to run COSMO in SP. They are listed in detail in Table 3. Those which are critical for the model not to crash in SP are emphasized.

3.2.1. Critical Epsilons

Epsilons are either range- or digit-limited, as established in Section 2.3. To account for this, we have introduced the two precision-dependent parameters *repsilon* and *rprecision*. They are defined in *data_parameters.f90* as shown below. Their magnitudes in SP/DP are $10^{-30}/10^{-300}$ and $10^{-7}/10^{-16}$, respectively.

The values chosen for epsilons in COSMO are typically between `repsilon` and `rprecision` in SP, for instance 10^{-15} or 10^{-8}. Thus they are big enough for range-limited cases, but might vanish in digit-limited contexts. Therefore, in range-limited cases, i.e. mostly in divisions, problems occurred in places where there had not been an epsilon previously, and we have had to add epsilons, predominantly in divisions where the divisor vanishes in SP. In digit-limited cases, on the other hand, an epsilon was usually already present in places where problems in SP occured, but with a too small magnitude. Thus, we have usually had to identify the respective epsilon parameter and limit is's magnitude to `rprecision` in SP using the `MAX()` function. Additionally, in some places, instead of a parameter a hard-coded epsilon has been used, e.g. 10^{-30}, usually in divisions. We have replaced those by epsilon parameters. The chosen magnitude of these hard-coded epsilons has usually not been critical in SP, except for one case where 10^{-50} was used.

In most cases, epsilon-related problems in SP manifested themselves in model crashes or deadlocks, i.e. were rather easy to recognize and track down. In one case where an epsilon vanished in SP, however, the implications were much more subtle, namely only impacted the model physics, and the problem was accordingly much more difficult to identify and resolve. This case is described in detail in Section 4.2.1.

data_parameters.f90

```
161   REAL (KIND=ireals), PARAMETER  ::                                  &
162        repsilon = 1.0E8_ireals*TINY(1.0_ireals),                     &
163                !
164                ! Very small number near zero.
165                ! To be used mainly to avoid division by zero, e.g.
166                ! eps_div = repsilon ; x = y / MAX(z,eps_div) ! for z >= 0.
167                ! Note that the factor 1.0E-8 has been chosen rather
168                ! arbitrarily to get some distance to zero to account
169                ! for the magnitude of the dividend, which might be 1.0E5
170                ! in case of pressure, for instance.
171                !
172
```

File	Modifications
data_obs_lib_cosmo	**limit `epsy` to `rprecision` in SP**
data_soil	new epsilons: ∘ `eps_div` - avoid DBZ ∘ `eps_soil` - threshold for computations ∘ `eps_temp` **- check if temperature below zero**
data_turbulence	replace `epsi` by `eps=repsilon`
near_surface	remove `zepsi`, use `repsilon` instead
numeric_utilities	∘ remove local epsilons (`zeps`) ∘ introduce module-wide variable `eps_div` instead
numeric_utilities_rk	∘ remove `zeps` ∘ new epsilons `eps_div` and `eps_adv`
organize_data	add definition of `epsy`
pp_utilities	∘ rename `eps` to `eps_conv` ∘ **reformulation** `m2s**4/m3s**3` →`m2s*(m2s/m3s)**3`
src_gscp	∘ replace `zeps` by `repsilon` where used as small number ∘ **reformulation** `m2s**4/m3s**3` →`m2s*(m2s/m3s)**3`
src_lheat_nudge	replace `epsilon`/`epsy` by `epsy` from *data_obs_lib_cosmo.f90*
src_obs_cdfin_print	use `epsy` from *data_obs_lib_cosmo.f90*
src_obs_cdfin_util	use `epsy` from *data_obs_lib_cosmo.f90*
src_obs_rad	add `ireals` to `REAL()`
src_output	replace `EPS` by `repsilon`
src_setup	add output `RUNNING IN DOUBLE/SINGLE PRECISION`
src_soil	replace `zepsi` by `eps_soil` from *data_soil.f90* or by `repsilon`
src_soil_multlay	replace `zepsi` by `eps_*` from data_soil
turbulence_tran	replace `epsi` by `eps` (both from *data_turbulence.f90*)
utilities	∘ **replace hard-coded `1E-50` by `repsilon`** ∘ **overload `check_field_NaNs()` for SP/DP**

TABLE 3: Detailed overview over conducted all changes, except those related to `ireals` declarations and the mixed precision (MP) radiation. Critical modifications are emphasized. These are necessary for the code to compile and run in SP, whereas the others can be considered code cleanup. Note that a number of epsilons added to divisions to avoid DBZ are not listed explicitly in this table.

data_parameters.f90

```
172        rprecision = 10.0_ireals**(-PRECISION(1.0_ireals))
173                !
174                ! Precision of 1.0 in additions/subtractions.
175                ! To be used for instance to check equality of reals, e.g.
176                ! eps_fpn = rprecision ; IF (ABS(a-b) < eps_fpn) equal=.true.,
177                ! or to increase the magnitude of an epsilon only in SP, e.g.
178                ! epsilon = MAX(1.0E-8_ireals,rprecision) when 1E-8 is too small
179                ! but the value should stay the same in DP.
180                !
181                ! Approximate magnitudes:
182                ! -----------------------
183                !
184                !       | repsilon | rprecision
185                !   ----+----------+------------
186                !    SP |  1.0E-30 |   1.0E-7
187                !   ----+----------+------------
188                !    DP | 1.0E-300 |  1.0E-16
189                !
```

3.2.2. Non-Critical Epsilons

Many of the epsilon-related changes listed in Table 3 are not strictly necessary to run COSMO in SP, but we have taken the opportunity to do some code cleanup, as the use of epsilons in the model is far from consistent, which is no surprise considering the large number of different people that have contributed it. Whereas in some parts of the code, such as the assimilation, centrally defined epsilons used by multiple source modules already exist, in other parts epsilons are defined very locally, i.e. on the subroutine level. In some cases the same epsilon parameter has been defined many times per file. This might be appropriate in special cases, such as utility routines, which should be as self-contained as possible. In most cases, however, one single definition of an epsilon per file is a much better and cleaner solution.

Soil Model

In the soil model (*src_soil.f90, src_soil_multlay.f90*) the same epsilon *zepsi* has been used for various purposes. We have replaced *zepsi* by the three new variables `eps_soil`, `eps_div`, and `eps_temp` Their definitions and descriptions are shown in the following code excerpt.

data_soil.f90

```
249   REAL  (KIND=ireals), PARAMETER :: &
250
251     ! Avoid division by zero, e.g. x = y / MAX(z,eps_div).
252     eps_div  = repsilon       , &
253
254     ! Threshold for various computations in soil model (former zepsi).
255     eps_soil = 1.0E-6_ireals  , &
256
257     ! Small value to check if temperatures have exceeded a fixed threshold
258     ! such as the freezing point.  In double precision (16 decimal digits)
259     ! a value as small value such as 1.0E-6 can be used. In single
260     ! precision (6-7 decimal digits), however, the value has to be larger
261     ! in order not to vanish. The current formulation is save for
262     ! temperatures up to 500K.
263     eps_temp = MAX(1.0E-6_ireals,500.0_ireals*EPSILON(1.0_ireals))
```

Assimilation

In the assimilation, there is one general-purpose epsilon variable called `epsy`, which is centrally defined in *data_obs_lib_cosmo.f90* and used in the files listed below.

data_obs_lib_cosmo.f90

```
 87   REAL    (KIND=ireals)        , PARAMETER  :: &
 88     epsy   = 1.0E-8_ireals    ,&! commonly used very small value > 0
```

- organize_assimilation
- src_correl_cutoff
- src_gather_info
- src_lheat_nudge
- src_mult_local
- src_mult_spread
- src_nudging
- src_obs_cdfin_blk
- src_obs_cdfin_comhead
- src_obs_cdfin_gps
- src_obs_cdfin_mult
- src_obs_cdfin_org
- src_obs_cdfin_print
- src_obs_cdfin_sing
- src_obs_cdfin_util
- src_obs_cdfout_feedobs
- src_obs_print_vof
- src_obs_proc_air
- src_obs_proc_aof
- src_obs_proc_cdf
- src_obs_processing
- src_obs_proc_util
- src_obs_use_org
- src_sfcana
- src_sing_local
- src_sing_spread

As the vast number of files suggests, `epsy` is used in a large variety of contexts, both precision-limited and algorithmic, and even physical. It's magnitude of 10^{-8} is a well thought-out compromise between all those use cases. While this works well in DP, we have run into problems in SP. For example, in one place, `epsy` is used in an abort criterion of an iteration. As it is digit-limited in this context, the minimal relative magnitude required is 10^{-7}. Because `epsy` is smaller (10^{-8}), it vanishes and the loop never terminates. We have fixed this by limiting `epsy` to `rprecision` in SP in-place.

src_gather_info.f90

```
4168       ELSEIF (    MAX( zwts1t1,zwts1t2 ) &
4169                 >= MAX( zwts0t1,zwts0t2 )+MAX(epsy,rprecision)) THEN
 ...
4172       ELSEIF ((    MAX( zwts1t1,zwts1t2 ) &
4173                  >= MAX( zwts0t1,zwts0t2 )-MAX(epsy,rprecision)) &
4174         .AND. (zwts1t2 > zwts0t2+epsy)) THEN
```

There are, however, many similar places in the code where `epsy` is not used in a robust way and might thus potentially cause problems. Globally limiting `epsy` to `rprecision` in SP is not a solution, because `epsy` is also used in physical contexts. Therefore, increasing it's magnitude in SP would have undesired impacts on the model physics. The many different contexts in which `epsy` is used, along with it's vast number of occurrences, suggest it might be advisable to replace it by several purpose-specific epsilons, analogous to our epsilon implementation in the soil model. However, such a clean implementation which would be robust in SP has not been done yet. We have only limited `epsy` to `rprecision` in-place in some critical cases, such as the one shown above, as a preliminary solution.

3.2.3. Optimized Formulas

We have conducted some reformulations of critical formulas to increase their accuracy and stability in SP, i.e. to avoid large error growth and model crashes. In addition, we have tested two reformulations for the numerical error they introduce in SP.

Critical Reformulations

There is one formula in the code which consitently caused the model to crash in SP in it's previous formulation (commented out in the code below). It occurs once in *pp_utilities.f90* and twice in *src_gscp.f90*.

pp_utilities.f90

```
3094 !    zn0s = 13.50_ireals * m2s**4 / m3s**3
3095      zn0s = 13.50_ireals * m2s*(m2s/m3s)**3
```

The values of `m2s` and `m3s` are on the order of 10^{-6} and 10^{-10}, respectively, plus/minus 2 orders of magnitude, i.e. both are already very small numbers before they are raised to higher powers. However as they are, they become minuscule, and `m3s**3` might even vanish in SP when it becomes $< 10^{-38}$, which causes FPEs due to DBZ. A simple reordering of the terms of the formula resolves the problem. In the new formulation the dividend and the divisor are both moderately small instead of minuscule numbers, as is the result of the division, and the third power does not cause any more issues.

Potential Precision Loss

Two potentially critical formulations in *src_radiation.f90* have been identified in previous work (Despraz and Fuhrer, *pers. comm.*). Both can be formulated in a different way, which might improve numerical accuracy. The first formula involves a sine and cosine of a variable. In the original code, the cosine is not computed directly from the operand, but from the sine, which saves an extra cosine array. However, computing it directly from the operand, without the intermediate sine, might be numerically more precise.

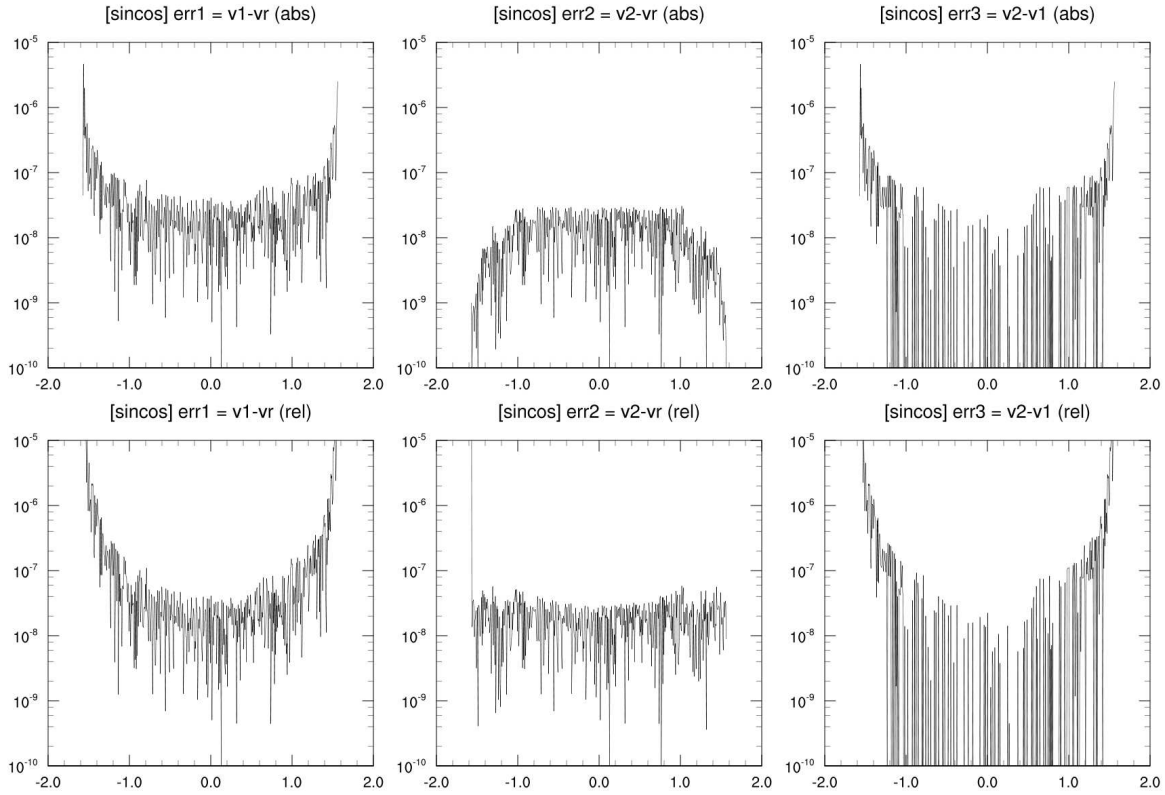

FIGURE 2: Numerical error in SP compared to $cos_x = cos(x)$ in DP of $cos_x = \sqrt{1 - sin(x)^2}$ (left) and $cos_x = cos(x)$ (center), as well as the difference between them (right). The difference corresponds to the gain in precision when substituting $cos_x = \sqrt{1 - sin(x)^2}$ by $cos_x = cos(x)$. The top panels show the absolute error, the bottom panels the relative error.

src_radiation.f90

```
1006        zsinphi  = SIN (degrad*(90.0_ireals-ABS(pollat)))
1007        zcosphi  = COS (degrad*(90.0_ireals-ABS(pollat)))
1008      ELSE
1009        zsinphi  = SIN(rlat(i,j) )
1010        zcosphi  = COS(rlat(i,j) )
1011      ENDIF
1012 !!     zcosphi  = SQRT(1._ireals-zsinphi**2)
 ...
2140             DO  i = 1, ie_tot
2141               zsinphi(i)      = SIN (rlattot(i,js))
2142               zcosphi(i)      = COS (rlattot(i,js))
2143             ENDDO
```

The second formula involves two formulations of the form $\left(1 - a^2\right)$, which can be reformulated to $(1 - a)(1 + a)$. Eliminating the square might be favorable in terms of numerical precision.

src_radiation.f90

```
5959 !!      pa1f(j1,j2) = ztau*(1.0_ireals-(zrho**2)) &
5960 !!                         *(1.0_ireals/(1.0_ireals-(zrho**2)*(ztau**2)))
5961       pa1f(j1,j2) = ztau*(zrho-1.0_ireals)*(zrho+1.0_ireals) &
5962                           /((zrho*ztau-1.0_ireals)*(zrho*ztau+1.0_ireals))
```

To assess the gain in precision in SP of the supposedly better formulations, we have computed the error of both formulations in SP relative to the better supposedly formulation in DP for all possible values. The errors

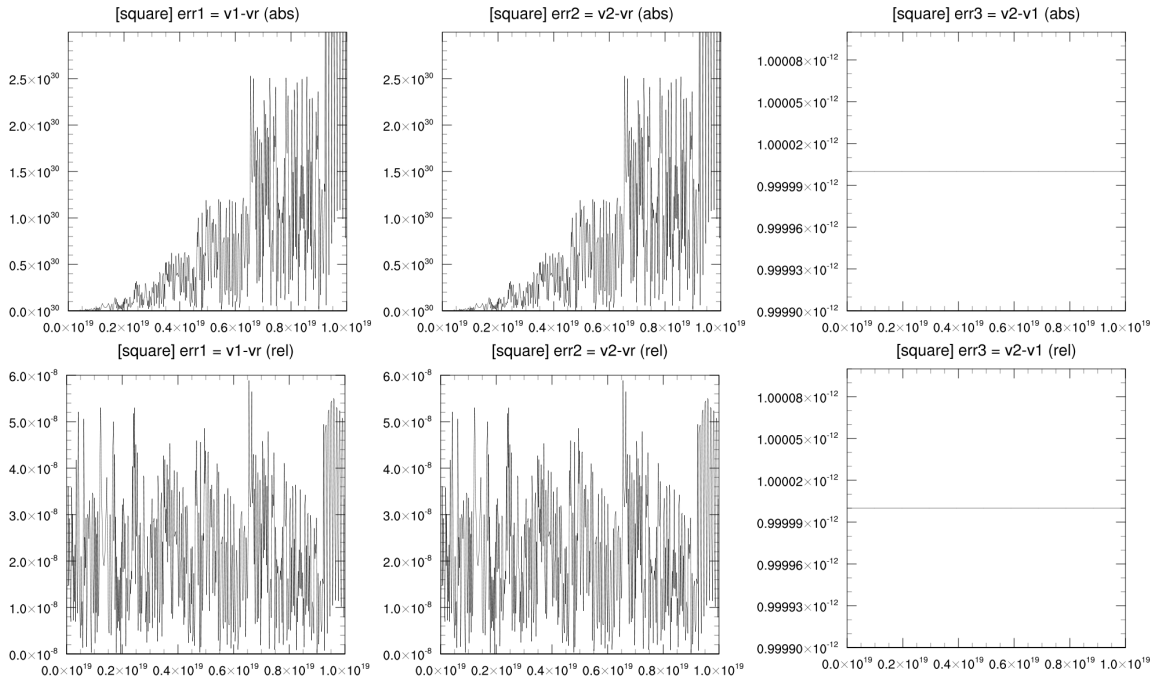

FIGURE 3: As in Figure 2, but for $1 - a^2$ (left) and $(1 + a)(1 - a)$(center) in SP relative to $(1 + a)(1 - a)$in DP. In this case, both formulations are totally equivalent with respect to precision.

in SP relative to DP, as well as their difference, are shown in Figures 2 and 3. The left and center panels show the error in SP of the old and new formulations, respectively. The difference between them in the right panels corresponds to the gain of using the supposedly better formulation. In case of the first formula, this gain is significant, which is why we have replaced the old formulation in the code. In case of the second formula, on the other hand, there is no gain, i.e. both formulations perform equally well in SP.

3.3. Mixed Precision Radiation

The only part of the code where substantial modifications are necessary to run it in SP is the radiation, which in it's current form has to run partly in DP regardless of the model's WP. It should be noted that we have not proven the algorithms in the radiation code to strictly require DP but rather have simply not succeeded in finding the critical modifications required to enable SP also for the radiation code.

The subroutine structure of *src_radiation.f90* is shown in Figure 4. The subroutines always running in DP are highlighted in yellow, those that are critical in orange. Figure 5 shows the dataflux between the main radiation subroutines, as well as that between them and the radiation data module, again with the critical and the DP subroutines highlighted. The dataflux by argument arrays at the interface of the WP and the DP part of the code (i.e. the calls to `fesft()` and `opt_th/so()`) is organized in such a way that the precision conversion of the `IN`-arrays is handled by the calling, and that of the `OUT`- and `INOUT`-arrays by the called subroutines[5].

The critical subroutines that only work in DP are the inversion routines `inv_th/so()`, along with their sub-subroutines `coe_th/so()`. Although `fesft()` works fine in SP, we have included it in the DP-part of the radiation because `inv_th/so()` are called so often by `fesft()`. If the conversion to DP and back were done on each of these calls, the model would be slowed down considerably. The second pair of subroutines called by `fesft()` are `opt_th/so()`. We have chosen to run them in WP despite their calling subroutine being run in DP, although this requires conversion of all argument arrays to WP and back on all calls. These conversions, however, do not effect the runtime as those on calls to `inv_th/so()` would, and the gain in terms of code simplicity is substantial. All arrays from *data_radiation.f90*, except for `cobi`, `coali`, `cobti`, `coai`, `planck`, `solant`, `zketypa`, and `zteref`, are only used in `opt_th/so()`[6]. Running `opt_th/so()` makes any DP

[5]Note that conversion is only done if necessary, i.e. if the model is run in SP. If the WP is DP, direct assignment is sufficient and the conversion is omitted.

[6]They are also used in `init_radiation()`, but this subroutine is always run in WP anyway.

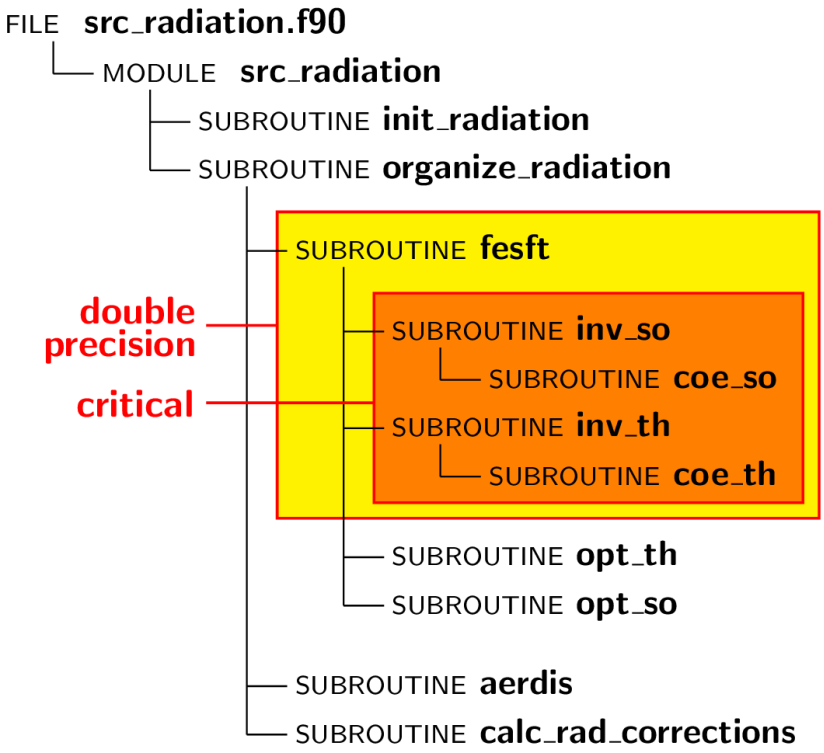

FIGURE 4: Structure of the source file *src_ radiation.f90*. The subroutines are nested according to how they call each other, the called subroutines indented with respect to the caller. The orange box encloses the subroutines that are critical with respect to precision, i.e. need to be run in DP regardless of the WP. The DP part has been expanded to what is enclosed by the yellow box due to performance considerations.

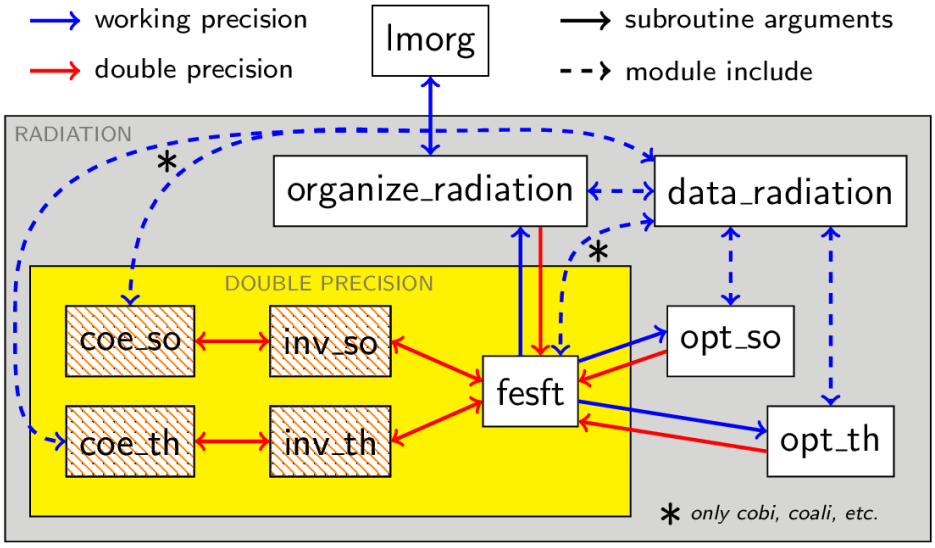

FIGURE 5: Data flux between the subroutines of the MP radiation. The DP part is enclosed by the yellow box, and the critical subroutines that only work in DP are emphasized by orange stripes. Solid arrows show data flux as array arguments, and dashed arrows show data exchange through the shared data module. Data flux happens in WP along blue arrows and in DP along red arrows. The precision transformation of `INOUT`- and `OUT`-arrays is handled by the called routine, wherease the handling of `IN`-arrays is left to the caller. Module data is only used by WP-subroutines, with the exception of the arrays `cobi`, `coali`, `cobti`, `coai`, `planck`, `solant`, `zketypa`, and `zteref`. The precision transformation of these arrays is handled by the DP subroutines that use them.

declarations of arrays in *data_ radiation.f90* unnecessary and thus restricts the changes to *src_ radiation.f90*. The three exceptions are handled by the DP subroutines using them.

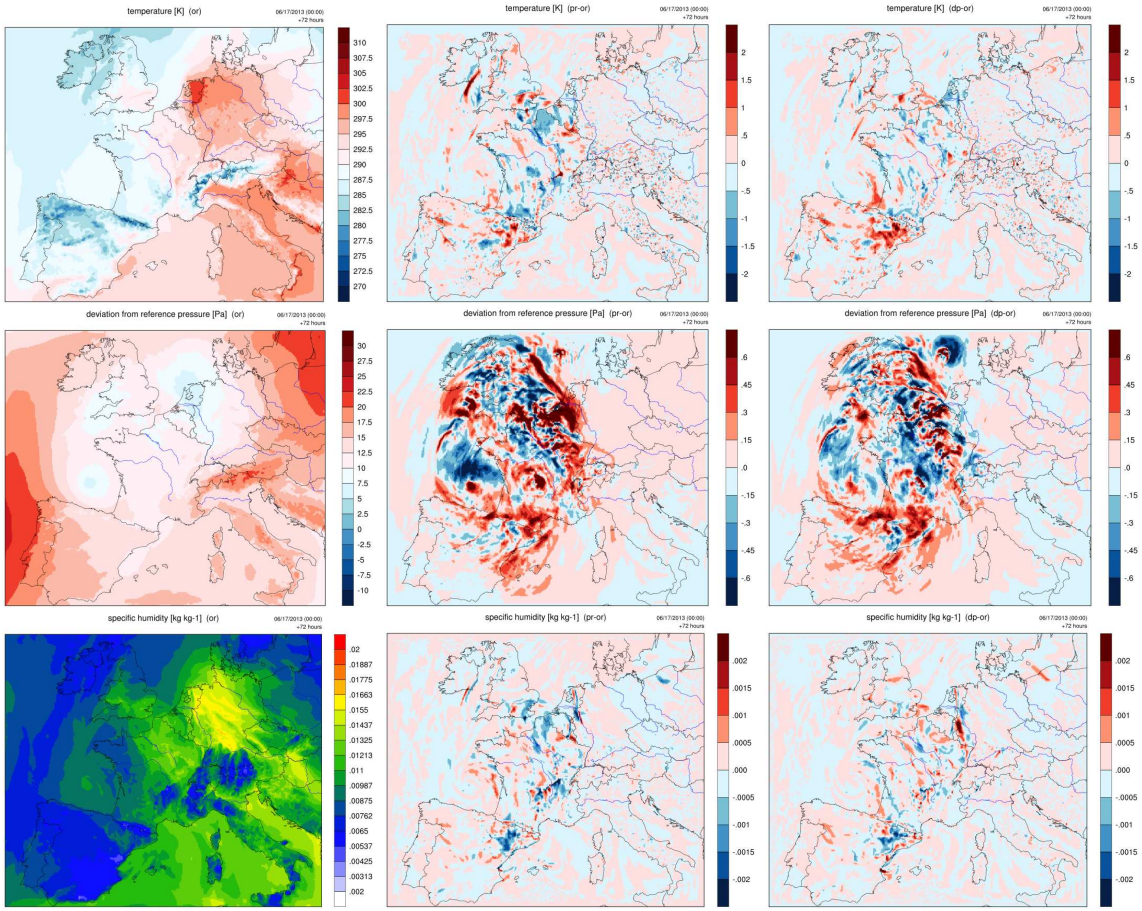

FIGURE 6: Results of the perturbation sensitivity experiments for temperature (top), pressure deviation (center), and specific humidity (bottom). The left panels show the absolute fields of **OR**, those in the center the deviations **PR-OR**, and those on the right **DP-OR**on the lowermost model level after 72h. The deviations **PR-OR** are comparable in magnitude to those of **DP-OR**.

4. Experiments and Results

To asses the performance of the new code, we have chosen a two-step approach. First, we conduct sensitivity experiments to test it from a numerical point of view. Second, we test it's performance against observations.

4.1. Sensitivity Experiments (COSMO-7)

To validate the new code from a numerical standpoint, we have conducted two major sets of sensitivity experiments. In a first step, we validate it in DP by simulating the effect of the additional `ireals` declarations. In a second step, we assess its performance in SP. The aim of these sensitivity experiments is to get a general feeling for the magnitude of the deviations between various code versions in a realistic setup, without aiming for meteorological representativeness. We use data from a period from last summer (June 17-19 2013). The model setup corresponds to MeteoSwiss operational COSMO-7 setup with a lead time of 72h.

4.1.1. Perturbation Experiments

The new code in DP, subsequently referred to as **DP**, will replace the original code (**OR**). Therefore, it is of high importance to make sure it yields results of equal quality. There are apparent differences between the results of the new and the original code. Considering the code changes we have carried out, those differences should be dominated by the effect of the correct type declaration of previously erroneously typed FPNs documented in Section 3.1. These may be seen as a random perturbation of relative magnitude $\mathcal{O}\left(10^{-7}\right)$. To assess the impact of such perturbations on the model physics, we have developed a new code (**PR**) based on

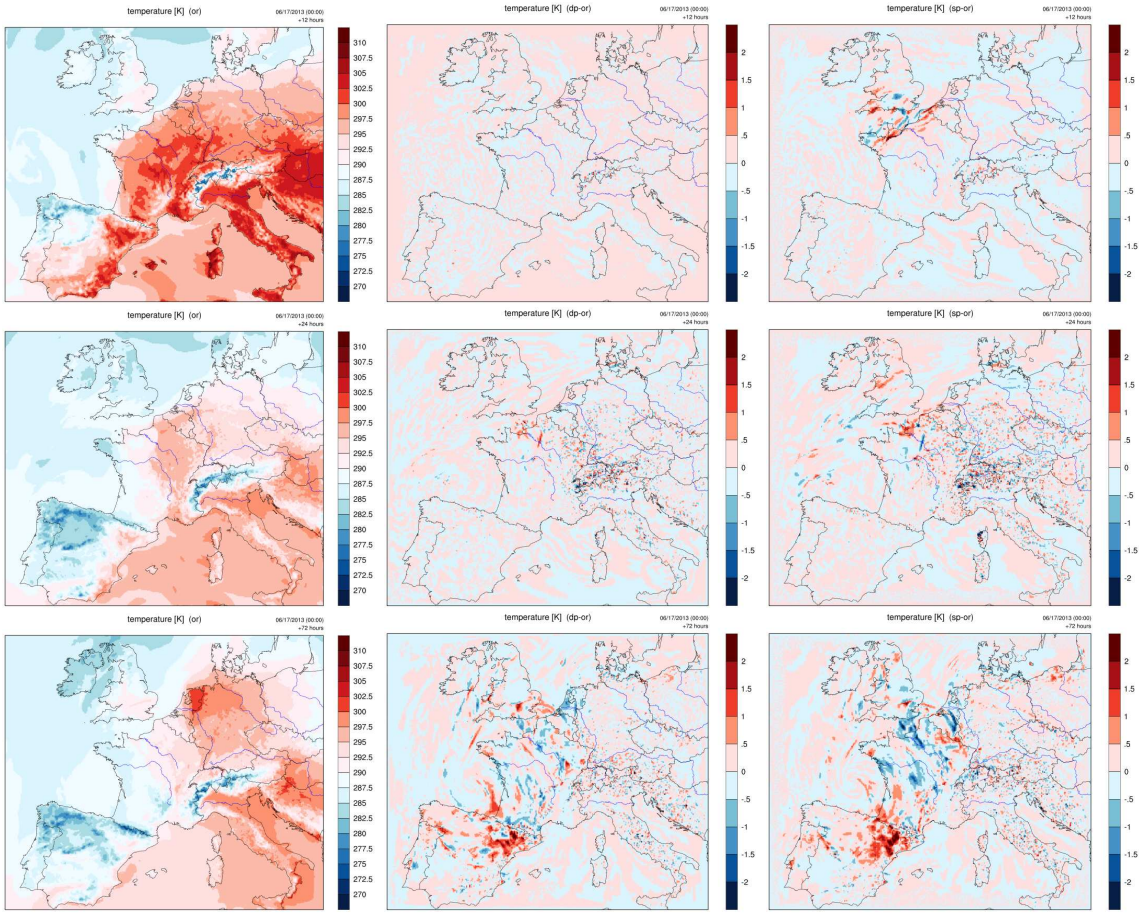

FIGURE 7: Results of the SP sensitivity experiments. Shown is temperature on the lowermost model level. Shown are the absolute field of **OR** (left), the deviations **DP-OR** (center), **SP-OR** (right). The lead times are 12h (top), 24h (middle), and 72h (bottom). The error growth is faster in **SP** than in **DP** with respect to **OR**, but the difference between **DP-OR** and **SP-OR** is already much smaller after 24h, and is gone after 72h, when they are of comparable magnitude.

the reference code, where all prognostic fields are randomly perturbed at every time step by a factor on the order of 10^{-7}. We run our test case with the three code versions **OR**, **DP**, and **PR** and compare the differences after 72h. Figure 6 shows the results of these experiments for temperature (top), pressure deviation (center), and specific humidity (bottom). Shown are the absolute fields of **OR** (left) as well as the differences **PR-OR** (center) and **DP-OR** (right) on the lowermost model level after 72h. The difference plots show perturbations of comparable magnitude for all three variables. These results confirms that the observed differences between **DP** and **OR** are mainly the result of the additional type declarations.

4.1.2. Single Precision Experiments

To assess the quality of the simulations with the new code in SP (**SP**), we run our test case with **OR**, **DP**, and **SP** and compare the fields after 12h, 24h, and 72h. The results are shown in Figures 7 and 8 for temperature and pressure deviation, respectively, after 12h (top), 24h (center), and 72h (bottom). After 12h, the deviations of **SP** from both **DP** (not shown) and **OR** are clearly larger than **DP-OR**. After 72h, however, the deviations between all model versions are of similar magnitude. No systematic biases are observed. We can conclude that perturbations due to numerical truncation errors seem to be slightly larger in **SP**, but physical error growth rapidly dominates. We have not found any indications that **SP** does not perform as well as **DP** in any of the experiments or variables analyzed.

4.2. Verification (COSMO-2)

The test which a NWP model eventually has to pass is verification against observations. From a numerical

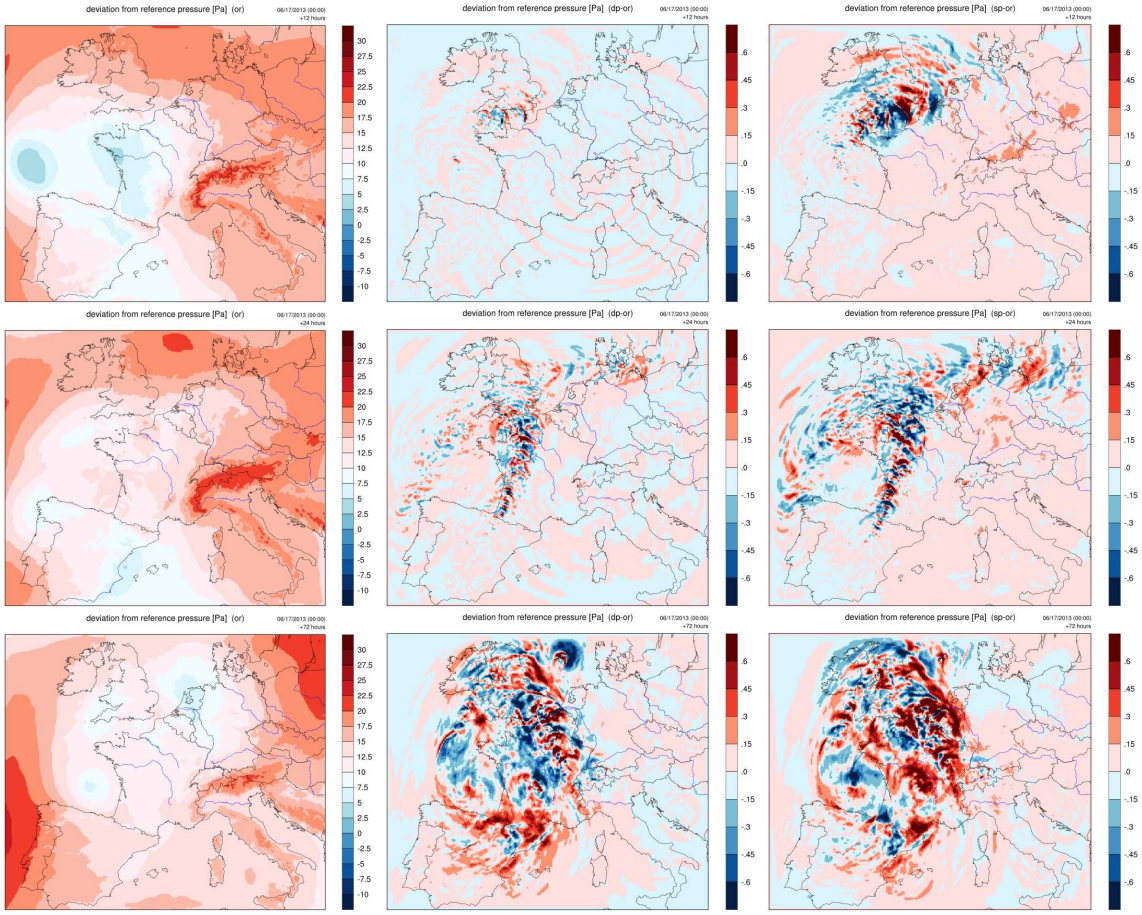

FIGURE 8: As Figure 7, but for pressure deviation.

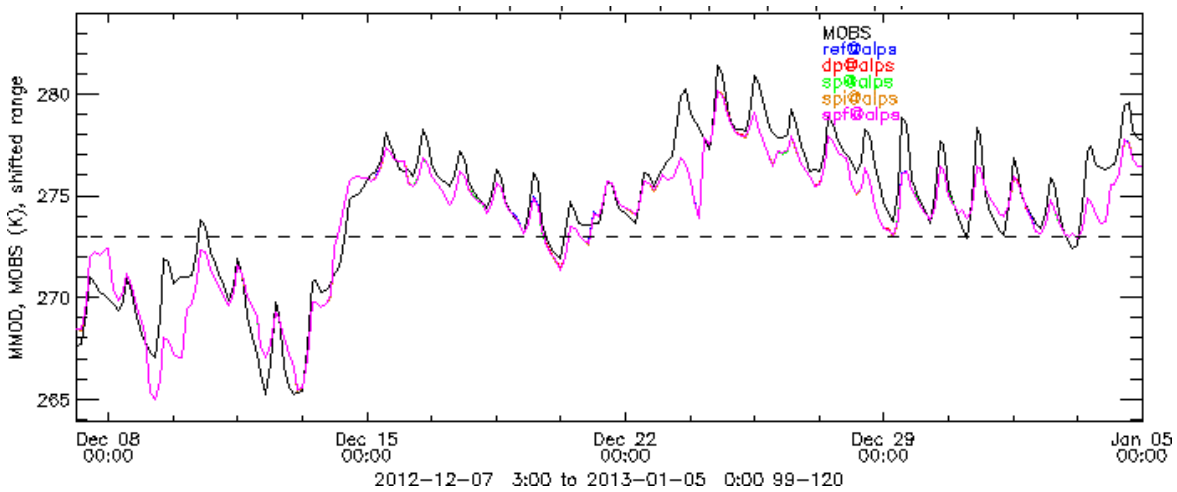

FIGURE 9: Verification against observations (black line) of OR (blue), DP (red), and SP (green, orange, purple) line). Shown is the mean 2m temperature for forecast day 5. The model forecoasts differ from the observations, but not at all from each other.

point of view, this test is far less restrictive than the sensitivity experiments presented in the previous section, as it allows for much larger deviations from the reference simulation. The verification runs are much more meteorologically representative, though, as a larger number of different situations occur during two months than during 72h, especially as both summer and winter are tested. Therefore it is much more likely for bugs in SP to show up in these runs than in those with the previous setup. The verification is done in the COSMO-2 setup. Simulations with lead time +120h are started every 24h during one month in both summer

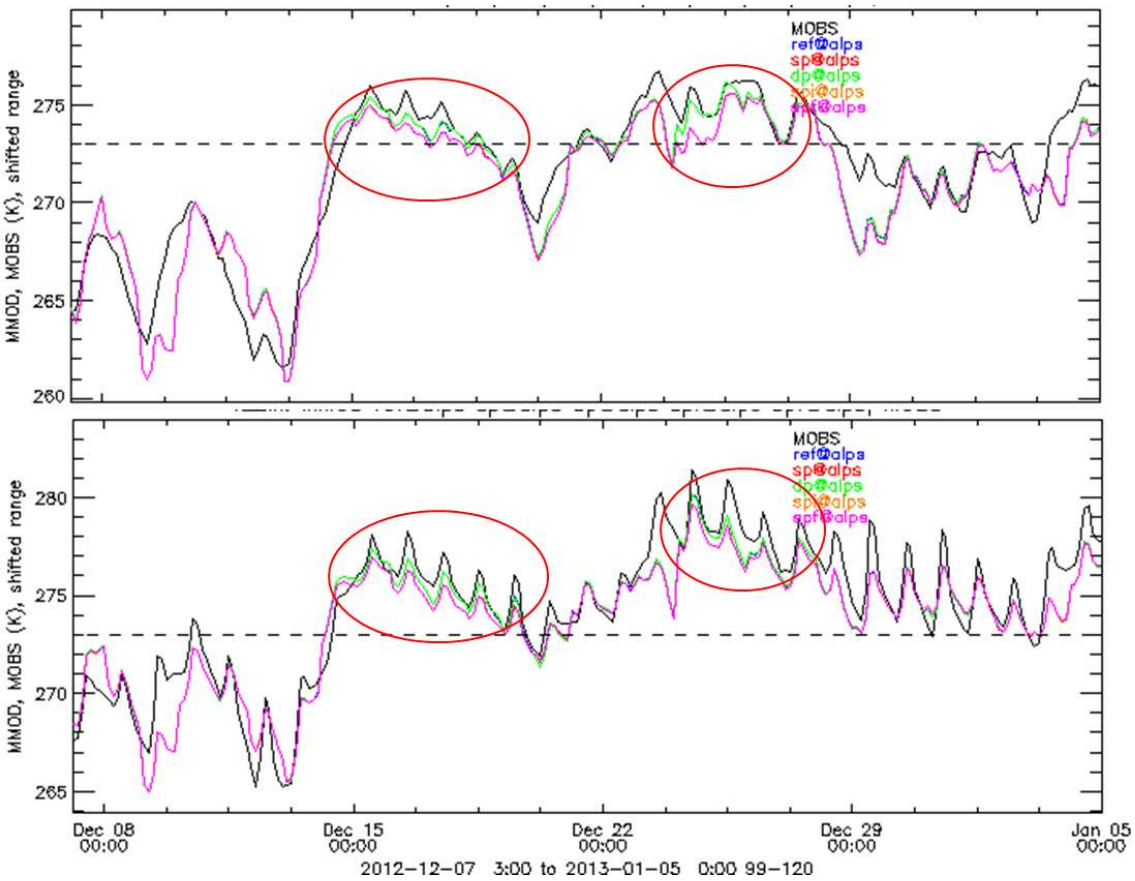

FIGURE 10: Verification against observations (black) of the code with the winter bug in DP (green) and SP (red, orange, purple). Shown are mean dew point temperature (top) and absolute temperature (bottom) for forecast day 5. The two periods where **SP** differs significantly from **DP** are emphasized by red circles. These differences have been caused by a bug in the soil model related to the melting of snow.

(August 2012) and winter (December 2012) and the results statistically compared to ground observations (up to +120h) and soundings (up to +72h).

The results of the **SP** and **DP** simulations are indistinguishable in most of the graphical products generated by the standard verification package. As an example, Figure 9 shows a time series of temperature on forecast day five. The forecasts (colored) are clearly distinguishable from the observations (black), however not from each other. We can conclude from the results that all code version (**OR**, **DP**, **SP**) perform equally well and are meteorologically not distinguishable.

4.2.1. The Winter Bug

In contrast to the final version, the first working **SP** code yielded ambivalent verification results. In summer, the performance of **SP** was comparable to that of the other codes. In winter, however, it was clearly inferior. Time series of absolute and dew point temperature showed significant deviations of **SP** from **DP** and **OR**, as shown in Figure 10 with the respective periods emphasized. Plots of the fields during these periods revealed strong temperature anomalies which quickly grew to large size by advection, with the source regions fixed in space, as shown in Figure 11 (top) for temperature at +96h (left) and +120h (right). On first sight the anomalies seemed to be spatially related to topography in some way. Investigation of various fields, however, eventually hinted towards differences in snow cover, as the source regions of the anomalies corresponded to the margins of snow covered regions. This led us to inspect the soil model code. We eventually identified the bug in *src_soil_multlay.f90*. It had been caused by an epsilon used to check whether a temperature has a certain minimal distance to the freezing point. We have replaced the epsilon variable `zepsi` by `eps_temp` in the critical places, as illustrated by the following code excerpt. The magnitude we have chosen for `eps_temp` is save up to temperatures of $500K$.

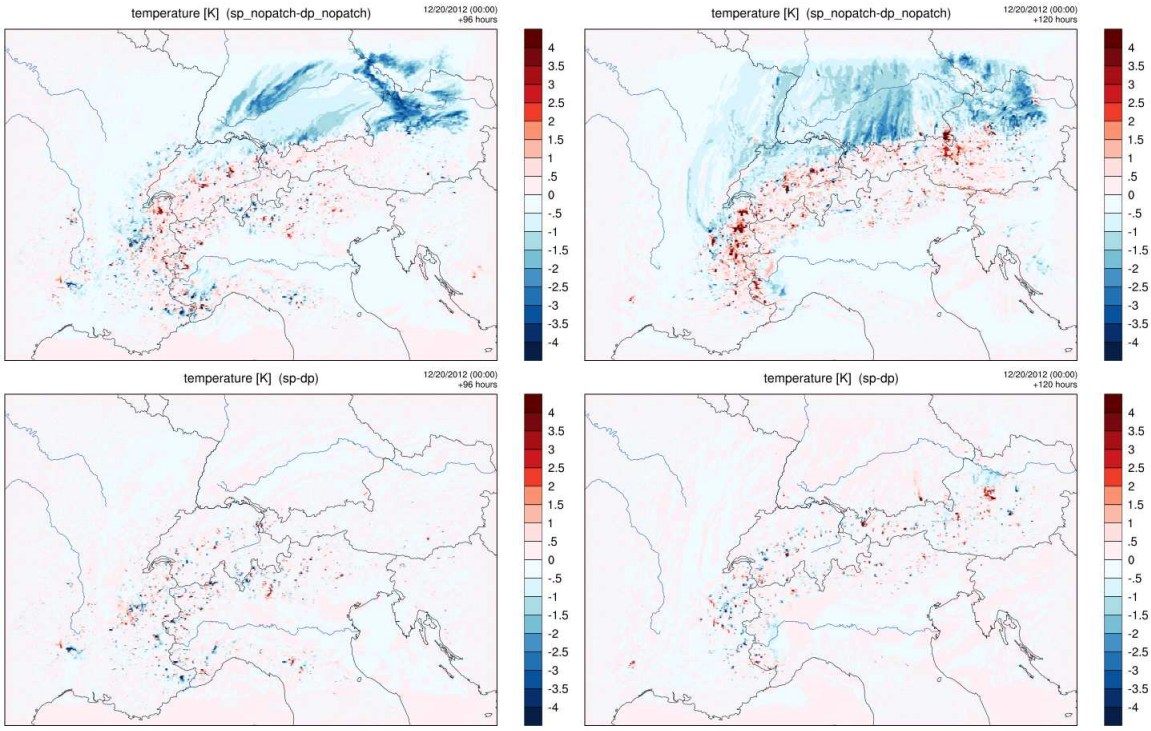

FIGURE 11: Difference in temperature between SP and DP with (top) and without (bottom) the winter bug after 96h (left) and 120h (right). (Top) The anomalies start to grow in specific places such as NW of Switzerland roughly parallel to the Jura mountains and along the Danube river valley, and from there grow by NE-ward advections. (Bottom) With the winter bug patch, the large deviations have disappeared, and only small, noisy signals over the alpine region remain.

src_soil_multlay.f90

```
 427   REAL     (KIND=ireals   ), PARAMETER ::  &
 428 !    zepsi  = 1.0E-6_ireals , & ! security constant
 429      eps_temp = MAX(1.0E-6_ireals,500.0_ireals*EPSILON(1.0_ireals))
 ...
3560 !          IF(t_so(i,j,kso,nnew).LT.(t0_melt-zepsi)) THEN
3561           IF(t_so(i,j,kso,nnew).LT.(t0_melt-eps_temp)) THEN
3562             zaa     = g*zpsis(i,j)/lh_f
```

This is a digit-limited case, i.e., the minimal relative epsilon in SP is 10^{-7}. The epsilon of magnitude 10^{-6} thus seems to be big enough at first sight. However, three significant digits are lost because of the order of magnitude of the temperatures (cf. Section 2.4), which increases the minimal magnitude of the epsilon by 10^3. `zepsi` therefore vanished in SP, introducing a hard-to-find bug. Replacing it by a new epsilon with a sufficiently large magnitude in SP solved the problem. The temperature fields from a model run with this patch are shown in Figure 11 (bottom). A comparison to the top panel shows that the large deviations have vanished, and that only a noisy signal of very localized deviations spread over the Alps remains, reminiscent of the deviations observed in the sensitivity experiments.

4.2.2. Timings

The major benefit of running the COSMO model in SP is a significant reduction in runtime. The mean runtime statistics with various model versions running on a Cray XE6 with AMD Magny-Cours processors are summarized in Table 4. Switching from DP to SP reduces runtime by almost 40%. This is substantial, as for example for an ensemble forecast the number of members could be increased by almost 70% on a given hardware.

OR	DP	SP	SP_i	SP_f
100.0%	102.0%	60.2%	56.2%	55.9%

TABLE 4: Mean runtimes of all sensitivity experiments relative to OR. SP_i and SP_f are SP with aggressive optimization at compilation (both without strict IEEE conformance, and SP_f additionally with less accurate library functions). The runtime of the most aggressively optimized code drops below 56%, i.e. it is almost twice as fast than the reference.

5. Discussion

5.1. Caveats

We have only tested the model configurations most important for MeteoSwiss. While this should cover the main parts of the model, we must emphasize that testing is required before using it in SP. It is in the user's responsibility to make sure his configuration of choice works in SP, especially if less common parts of the code are to be run. Critical in this respect is the assimilation due to the unresolved epsilon issue presented in Section 3.2.2.

The synthetic satellite image package RTTOV7 does not work in SP. This is why we have written wrapper subroutines for all RTTOV7 subroutines used in COSMO. They convert the input arguments from WP to DP, call RTTOV7 in DP, and convert back the output arguments.

5.2. Epsilon Recommendations

In this article, we have presented several epsilon-related pitfalls and some often-made mistakes in COSMO. To help avoid such problems in the future, some of which are critical to run the model in SP, we have compiled a few recommendation.

Use purpose-specific epsilon-parameters. Try to avoid multi-purpose epsilon parameters the magnitude of which is a compromise between various use cases. We recommend to consider the use of at least three distinct epsilon parameters.

- `eps_div`: Range-limited epsilon to avoid division by zero (DBZ).
- `eps_fpn`: Digit-limited epsilon for FPN comparisons.
- `eps_???`: Algorithmic epsilon(s) for all other purposes.

Insert digit-limited epsilons by multiplication, not by addition. Digit-limited epsilons are those the magnitude of which, relative to other involved numbers, is the important factor. This is predominantly the case in comparisons of FPNs. Inserting such epsilons by multiplication (`a < (1.0 + eps)*b`) considers this relative nature and is much more robust than the usual insertion by addition (`a < b + eps`).

Define epsilon parameters neither too locally, nor to globally. Neither should they be redefined in every subroutine in a file, nor should only a single epsilon be used throughout the whole model. A good compromise is usually one definition per file/module, or, if there is one, in a data module, which might be shared by multiple source modules. Do not hardcode the values of epsilons, but use the newly introduced precision-dependent parameters `repsilon` and `rprecision` to define range- and digit-limited epsilons, respectively.

Name epsilon parameters in a way that reflects their purpose. Especially algorithmic epsilons should be marked as such in order for them not to be confused with precision-limited epsilons, e.g. `eps_soil` instead

of a plain `epsilon`. Also clearly state the purpose of an epsilon parameter in a detailed comment next to it's definition.

Be aware of rounding error. Think about rounding error when writing new code and be aware of users running the COSMO model in reduced precision. Test your code in reduced arithmetic precision (single precision), if possible. If it runs and validates, it is highly likely to work correctly in higher precision arithmetic (double precision).

5.3. Summary and Conclusions

In this article, we have presented the modifications necessary to run the COSMO model in single precision (SP) instead of double precision (DP), and how this modified version of COSMO performs in both DP and SP. Three types of modifications are necessary. In a first step, all real type declarations (`ireals`) currently missing in the code are added to obtain a model running purely in DP. These are the changes which alter the results of DP-runs of the model (in a numerical sense). In a second step, various local modifications are introduced. These are mainly related to epsilons, i.e. additional epsilons to prevent model crashes in SP, or working precision-dependent modifications of the magnitude of some epsilons. For the latter purpose, we have introduced two precision-dependent parameters. Furthermore, a few formulas need to be reformulated to a numerically more precise form. The third step towards SP is a mixed precision implementation of the radiation, which in it's current form does not work in SP. A substantial part of the radiation is therefore always run in DP.

We have validated the code in two steps. First, sensitivity experiments have shown that the differences between the new code in DP and the reference code can be explained by the addition of the missing real kind declarations, and that, despite slightly faster error growth, the differences between the new code in both SP and DP from the reference do not show any systematic biases and are of the same order of magnitude for longer lead times. Second, validation against observations has shown that the forecast quality is unaffected by the switch from DP to SP. The gain of the switch to SP is a reduction of the model runtime to roughly 60% accompanied by a significant reduction of required memory.

Running COSMO with reduced precision should be a good choice for many applications. However, we must emphasize that our tests of the model in SP have been far from exhaustive. There might still be places in the code where problems occur in SP that have yet to be discovered. The elaborate documentation of the changes and problems we have presented in this article may serve as a guideline to cope with future problems with either previously untested or new code in SP.

References

[1] ANSI/IEEE 754 (2008). IEEE standard for binary floating-point arithmetic.

[2] Düben, P. D., McNamara, H., and Palmer, T. N. (2013). The use of imprecise processing to improve accuracy in weather and climate prediction. *J. Comp. Phys.*, in press.

[3] Govett, M., Middlecoff, J., and Henderson, T. (2010). Running the NIM next-generation weather model on GPUs. In *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, pages 792–796.

[4] Riedinger, K. (2011). Mixed precision im dynamischen kern eines numerischen wettervorhersagemodells. Bachelor's thesis, University Innsbruck.

[5] Skamarock, W. C., Klemp, J. B., Dudhia, J., Gill, D. O., Barker, D. M., Duda, M. G., Huang, X.-Y., Wang, W., and Powers, J. G. (2008). A description of the advanced research WRF version 3. NCAR technical note, National Center for Atmospheric Research, Boulder, CO, USA.