Response to Review Comment 1 (RC1)

Manuscript:	egusphere-2023-2129
Title:	A dynamic approach to three-dimensional radiative transfer in numerical weather prediction models: the dynamic TenStream solver v1.0
Authors:	Richard Maier, Fabian Jakub, Claudia Emde, Mihail Manev, Aiko Voigt, and Bernhard Mayer

We thank Anonymous Referee #1 for his or her comments on our manuscript, which we will respond to below. To structure our response, the referee's comments are printed on a gray background color, while our answers are displayed on ordinary white background.

This paper describes a method for 3D radiative transfer that could be computationally affordable enough to be used in high-resolution models. The idea of treating radiation more akin to dynamics is intriguing and as far as I know novel. The results presented are state-of-the-art in terms of speed-accuracy tradeoff (at least for 3D solvers) and potentially very significant for the advancement of NWP models, which are already configured at resolutions where 3D radiative effects are notable yet are currently ignored in all operational models.

My major comments are provided below and relate mainly to the computational aspects, which deserve more attention. Some of my questions may be adequate to address in the review and not in the paper, as it's already long (and concerned mainly with demonstrating the feasibility of the method - which it does excellently!), but a few clarifying sentences and providing absolute runtimes and/or measures of floating point operations in the paper would go a long way in informing the reader how fast dynamic tenStream potentially is, and whether it could be a real contender to operational radiation schemes outside of LES.

Besides this, I think the paper would really benefit if the authors tried to make it more concise by avoiding repetition and removing unnecessary words and sentences. The results shown are relevant but they are sometimes described in a very wordy manner.

Finally, the code does not seem to be actually available to download at current time which I understand is against GMD policy.

Other major comments:

1a. In general it's a bit difficult to fully understand the method (although Figure 3 does a good job at illustrating it) especially when it comes its implementation in code and its parallelism. The future tense used in L198-204 implies that the parallelism is not yet implemented. My understanding of dynamic TenStream would be something like this for a simplified 1D case:

```
! Downwelling flux; boundary condition

fd(1) = incsol

fd(2:nlev) = fd_prev_timestep(2:nlev)

! Gauss seidel incomplete solves, not parallelizable

for jiter in 1,niter
```

```
! Vectorization or other parallelism, array notation

fd(2:nlev) = T(1:nlev-1)*fd(1:nlev-1)
```

This would correspond to the radiative flows in individual grid boxes being computed concurrently i.e. in parallel within a single step of Fig 3, is this right?

Yes, your simplified 1D case does indeed illustrate the concept of the dynamic TenStream solver, although you use the Jacobi method instead of the Gauß-Seidel method to update the outgoing fluxes of the grid boxes. In contrast to the Gauß-Seidel method, this Jacobi method always uses ingoing fluxes from the previous time step to calculate the updated outgoing fluxes of a grid box. It would thus also allow for concurrent calculation of these outgoing fluxes. On the downside, information can only be propagated to the neighboring grid boxes in every single Jacobi iteration, leading to slow convergence despite high parallelizability.

This is why we have chosen to use the Gauß-Seidel method instead. In your simplified 1D case, this Gauß-Seidel method would look something like this:

```
for jiter in 1, niter
    for iz in 2, nlev
        fd(iz) = T(iz-1) * fd(iz-1)
```

In contrast to the Jacobi method you described, it uses updated ingoing fluxes wherever possible in the calculation of outgoing fluxes, leading to much faster convergence. When calculating fd(iz), for example, we can already use the value of fd(iz-1) determined in the very same iteration jiter. However, this implies that the Gauß-Seidel method does not allow for concurrent calculation of outgoing fluxes for all the grid boxes, as it would simply lead to the Jacobi method in that case: when doing the calculations for all the grid boxes in parallel, we would always have to use ingoing fluxes of the previous iteration instead of the current iteration.

In order to parallelize the Gauß-Seidel method, our idea is thus to apply parallelization to subdomains of the full 3D domain that are larger than an individual grid box. Within every one of these subdomains, the use of the Gauß-Seidel method would ensure that already updated ingoing fluxes are used in the calculation of outgoing fluxes wherever possible, speeding up convergence. Updates between different subdomains would only happen in between different calls of the radiation scheme. This treatment would represent a balance between convergence speed and parallelizability.

But as you correctly noted, parallelization has not yet been implemented into the dynamic TenStream solver by now.

1B. How should the reader interpret the reported speed numbers in terms of effective speed against operational radiation schemes? Is the 1D delta-Eddington reference based on efficient, vectorized code? It is unclear how efficient dynamic TenStream is or could be compared to widely used two-stream codes such as ecRad, which expresses parallelism across g-points, or the RTE+RRTMGP scheme which vectorizes the column dimension instead. Comparison to other schemes could be greatly facilitated by reporting absolute runtimes, or you could run one of the aforementioned schemes. Potential lack of parallelism and optimization in its current stage can be stressed explicitly and of course, even if dynamic tenStream is currently much slower than operational schemes then it's not a bad result considering full 3D solvers have until now been many orders of magnitudes more expensive. Finally, it could be very useful to report the number of floating point operations (whether absolute or relative to delta-Eddington) but may require a library such as GPTL to estimate, and is perhaps not necessary if the other aspects are clarified.

The relative numbers in Table 1 can indeed not be used to compare the speed of these solvers with respect to those in operational radiation schemes. However, providing such numbers was never the intention of this paper, as the dynamic TenStream solver is still in an early stage of development.

The main point we wanted to make in terms of speed was that a solver using incomplete solves is pretty fast by its design, as it only updates the fluxes in the radiative field a limited amount of times, which is much closer to the way 1D independent column approximations work, where you only update the fluxes of every grid box once every time the radiation model is called.

In order to provide a rough estimate of how fast the solver currently is, we performed this simple speed comparison to other solvers in libRadtran that are indeed not based on highly efficient code and not parallelized either.

We think that simply providing absolute runtimes instead would not really add any value, as these runtimes are highly dependent on the environment the code is executed in: the retrieval of the TenStream coefficients from the corresponding look-up tables is for example highly dependent on where these coefficients are stored. On top of that, the dynamic TenStream solver is not yet parallelized, making comparisons to highly efficiently written and parallelized solvers not particularly useful.

2. Can you discuss whether you see dynamic TenStream to be a potentially viable scheme for global or regional NWP models as they approach kilometer scale resolution? And on cost again: as these models currently use a very coarse radiation time step compared to the ones reported in the paper, such as 15 minutes (AROME 2.5 km regional model) or 1 hour (IFS, but 9 km so not yet km-scale), does this mean that dynamic TenStream would in fact incur a much bigger cost increase for such models than those given in Table 1, or does the coarser spatial resolution compared to LES mean that dynamic TenStreams convergence would still be adequate with relatively coarse radiation time steps?

As the radiative field changes much less rapidly at the lower resolutions used in global or regional scale NWP models, we would assume that also a much coarser radiation time step is needed to achieve comparable results as for the high-resolution test case presented in this paper. On top of that, performing more Gauß-Seidel iterations per radiation call does in fact not scale linearly with computational time, as the computational time is mainly determined by overhead such as retrieving the TenStream coefficients from the look-up tables when performing such a low number of iterations. For our test case, using 10 instead of 2 Gauß-Seidel iterations for example is less than two times more expensive. One could hence easily try to perform a bit more iterations per radiaton call if 2 Gauß-Seidel iterations would not be sufficient to run into proper convergence.

However, this is just speculation at this point in time. To really figure out how incomplete solves perform on the NWP scale, we will have to adapt the model for kilometer-scale resolutions and thoroughly test it, which is beyond the scope of this paper.

To clarify that the paper focuses on subkilometer-scale models for now, we have changed to title to "A dynamic approach to three-dimensional radiative transfer in subkilometer-scale numerical weather prediction models: the dynamic TenStream solver v1.0".

Minor comments:

Section 2.1. For the direct radiation, what is the advantage of having 3 streams in the independent x,y,z directions rather than two streams to/from the direction of the sun?

Finite volume algorithms such as the TenStream solver require the calculation of radiative fluxes for at least all the surfaces of the underlying grid boxes – for cuboids, which is the type of grid boxes used in the libRadtran library, that would add up to a total of six streams. Since direct radiation propagates into just one specific direction at every cuboid face, the number of streams can be further reduced to three. That, however, is the minimum amount of streams possible for direct radiation.

L114: Does TenStreams use of an external linear algebra library mean that its implementation is computationally efficient and exploits parallelism but dynamic TenStream currently does not, if so can the speed-up reported in Table 1 be improved further in the future?

Indeed, the use of PETSc allows the original TenStream solver to use computationally efficient methods to solve its system of linear equations. In addition to that, it is also parallelized. However, one of the main aims in the development of the dynamic TenStream solver was to get rid of complex libraries such as PETSc to allow for easier integration into operational models.

Besides that, we do not assume that the numbers in Table 1 will improve when parallelizing the dynamic TenStream solver, as they all refer to the single core performances of the corresponding solvers. Regarding multi-core performance, the TenStream solver is usually much more memory bandwidth limited than 1D delta-Eddington solvers are. As shown in Jakub and Mayer (2016), this leads to the original TenStream solver actually scaling worse to more cores than traditional 1D solvers do.

L114: Does PETSc run on GPUs? Do you think GPU acceleration is promising for (dynamic) tenStream?

Yes, PETSc does indeed run on GPUs. That being said, the Gauß-Seidel method used in the dynamic TenStream solver is a notoriously bad solver on GPU compute architectures. Other solvers such as the Jacobi method are better suited towards the high parallelization on GPUs and have been tested for the original TenStream solver using PETSc on GPUs. However, run times turned out to be only on par or just slightly better than when using CPUs.

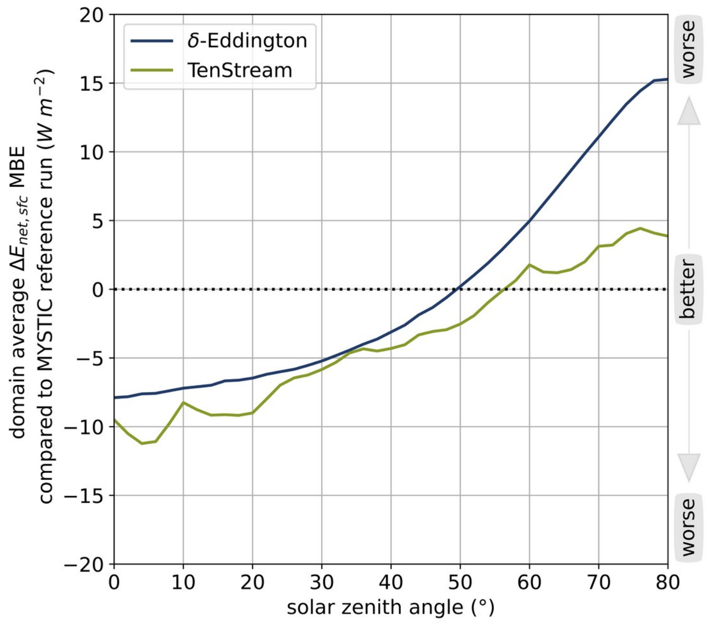
In addition to that, the computing time of the dynamic TenStream solver is mainly determined by CPU-based overhead such as the retrieval of the TenStream coefficients and not so much by the actual Gauß-Seidel solve. Hence, we do not assume a notable increase in speed by just performing the Gauß-Seidel iterations on GPUs.

L272. Has TenStream been evaluated across a wider range of solar zenith angles and is its performance sensitive to it?

Yes, the TenStream solver has been evaluated at a wide range of solar zenith angles and its performance is sensitive to it (Jakub and Mayer, 2016). Especially when considering small (sub)domains combined with high zenith angles, information has to be transported over multiple subdomains in case of parallelization, slowing down convergence as communication between different cores is required.

L474-495. Interesting, what is the reason for tenStream having a worse surface irradiance bias than delta-Eddington?

As Anonymous Reviewer #4 pointed out, the solar zenith angle that we have used in our calculations is very beneficial for the 1D delta-Eddington solver, as there are two different 3D radiative effects at the surface that cancel out for solar zenith angels around 45°. At large solar zenith angles, i.e., when the Sun is close to the horizon, 1D radiative transfer overestimates the net surface irradiances due to smaller shadow regions. In contrast to that, the lack of horizontal transport of diffuse radiation leads to 1D radiative transfer underestimating net surface irradiances at small zenith angles. Initially, we have not evaluated the time series for different zenith angles, which is why we did not give an explanation for that in the paper. For the revised version, we investigated that in more detail. The following figure shows the mean bias error in the net surface irradiance as a function of the solar zenith angle for both the 1D delta-Eddington solver and the original TenStream solver, evaluated for the very first time step in our time series:



It basically confirms that the surface irradiance bias at a solar zenith angle of 50° that we used in our evaluation is very beneficial for the 1D solver, although the TenStream solver performs worse than the 1D delta-Eddington solver at all solar zenith angles below 50°. However, the difference in the MBE between the solvers is very small, as the absolute MBEs for solar zenith angles below 50°

shown in Fig. 3 both result in relative mean bias errors of about -1% (not shown here). For higher solar zenith angles, we can however clearly see that the TenStream solver outperforms the 1D delta-Eddington solver.

L540-544. This is an example of probably unnecessarily detail and wordiness (4 lines of text to introduce a plot similar to one already shown)

You are absolutely right that Fig. 11 is introduced far too detailed. For the revised version, we significantly shortened this part as follows: “Before making a closing statement, let us also have a look at the results in the thermal spectral range shown in Fig. 11.”

References:
Jakub, F. and Mayer, B.: 3-D radiative transfer in large-eddy simulations – experiences coupling the TenStream solver to the UCLA-LES, Geosci. Model Dev., 9, 1413–1422, https://doi.org/10.5194/gmd-9-1413-2016, 2016