Twelve times faster yet accurate: a new state-of-the-art in radiation schemes via performance and spectral optimization

JAMES Journal of Advances in Modeling Earth Systems®

RESEARCH ARTICLE
10.1029/2023MS003932

Key Points:

- Restructuring and serial code optimization is performed for the ECMWF radiation scheme
- When combined with spectrally reduced gas optics, a speed-up of 12 is obtained compared to the operational scheme
- Cloud 3D radiative effects become affordable to compute in dynamical models

Correspondence to:
P. Ukkonen,
peterukk@gmail.com

Citation:
Ukkonen, P., & Hogan, R. J. (2024). Twelve times faster yet accurate: A new state-of-the-art in radiation schemes via performance and spectral optimization. *Journal of Advances in Modeling Earth Systems*, *16*, e2023MS003932. https://doi.org/10.1029/2023MS003932

Received 14 JUL 2023
Accepted 8 DEC 2023

Twelve Times Faster yet Accurate: A New State-Of-The-Art in Radiation Schemes via Performance and Spectral Optimization

Peter Ukkonen[1] and Robin J. Hogan[2,3]

[1]Danish Meteorological Institute, Copenhagen, Denmark, [2]European Centre for Medium-Range Weather Forecasts, Reading, UK, [3]Department of Meteorology, University of Reading, Reading, UK

Abstract Radiation schemes are critical components of Earth system models that need to be both efficient and accurate. Despite the use of approximations such as 1D radiative transfer, radiation can account for a large share of the runtime of expensive climate simulations. Here we seek a new state-of-the-art in speed and accuracy by combining code optimization with improved algorithms. To fully benefit from new spectrally reduced gas optics schemes, we restructure code to avoid short vectorized loops where possible by collapsing the spectral and vertical dimensions. Our main focus is the ecRad radiation scheme, where this requires batching of adjacent cloudy layers, trading some simplicity for improved vectorization and instruction-level parallelism. When combined with common optimization techniques for serial code and porting widely used two-stream kernels fully to single precision, we find that ecRad with the TripleClouds solver becomes 12 times faster than the operational radiation scheme in ECMWF's Integrated Forecast System (IFS) cycle 47r3, which uses a less accurate gas optics model (RRMTG) and a more noisy solver (McICA). After applying the spectral reduction and extensive optimizations to the more sophisticated SPARTACUS solver, we find that it's 2.5 times faster than IFS cy47r3 radiation, making cloud 3D radiative effects affordable to compute in large-scale models. The code optimization itself gave a threefold speedup for both solvers. While SPARTACUS is still under development, preliminary experiments show slightly improved medium-range forecasts of 2-m temperature in the tropics, and in year-long coupled atmosphere-ocean simulations the 3D effects warm the surface substantially.

Plain Language Summary A crucial step in simulating weather and climate is calculating how atmospheric radiation (shortwave radiation from the sun and terrestrial longwave radiation) interacts with the Earth's atmosphere and surface. The complexity of the underlying physics has necessitated making approximations in how radiative transfer is treated, such as assuming that radiation can only enter or leave a cloud through its top or base, thereby ignoring 3D effects. Even so, radiative transfer has historically been one of the computationally most demanding steps in making weather and climate simulations. Here we show that a state-of-the-art radiation code can be sped up threefold by using code optimization techniques that seek to maximize performance on modern processors. Combining this with a recent innovation that reduces the number of spectral computations required for accurate solutions, an order-of-magnitude increase in speed is obtained compared to the existing radiation scheme in a global weather model. Crucially, these improvements also make a radiation scheme that accounts for cloud 3D radiative effects fast enough to be used operationally. When included in global simulations, these 3D effects act to warm the lower atmosphere substantially.

1. Introduction

Atmospheric radiation is well-understood, but too complex to be solved in an exact manner in weather and climate models. That is, with the exception of the treatment of sub-grid cloud structure, very accurate solutions to atmospheric radiative transfer are available but too costly to use in dynamical models. This leaves its parameterization as an exercise in how to obtain as accurate broadband longwave and shortwave fluxes as possible with the least possible computational cost. For the spectral integration, the correlated-k-distribution method (CKD, e.g., Goody et al., 1989) has emerged as a leading solution. CKD is based on reordering the highly detailed absorption spectra of atmospheric gases by its optical properties into a cumulative probability function. Accurate spectral integration then becomes possible with only O(10^2–10^3) quadrature points—known as k-terms or g-points—compared with O(10^6–10^7) for line-by-line methods.

Despite the use of CKD, and considering the transfer of diffuse radiation only in the upward and downward directions (“two streams”), radiation computations are expensive enough that their temporal and/or spatial frequency is often limited. In high-resolution forecasts based on the Integrated Forecast System (IFS), a global numerical weather prediction (NWP) model developed at the European Centre for Medium-Range Weather Forecast (ECMWF), its radiation scheme ecRad is called every hour on a grid with roughly 10 times fewer columns than the rest of the model (Hogan & Bozzo, 2018). Such approximations are a source of uncertainty in large-scale models. In particular, 3D radiative effects by clouds are routinely ignored in weather and climate simulations, yet were estimated by Schäfer (2017) to be similar in magnitude to anthropogenic greenhouse gas forcing. (This does not mean they are as important for climate projections as it refers to the mean 3D effect in current climate—a bias that may be masked by model tuning or compensating biases—and not a change in 3D effects under climate change). Due to the spatial and temporal coarsening, ecRad is only a few percent of the total IFS runtime (Hogan & Bozzo, 2018), but radiation becomes more expensive for lower-resolution simulations where it must be called at a higher frequency relative to the model time step. For instance, in a coarse-resolution setup of the ECHAM climate model, radiation accounted for half of the runtime of the atmospheric model (Cotronei & Slawig, 2020).

The perceived expense of radiation schemes has led to attempts to replace them with faster and approximative neural network (NN) emulators (e.g., Chevallier et al., 1998; Krasnopolsky et al., 2008; Pal et al., 2019; Song & Roh, 2021), avoiding explicit spectral computations and typically predicting heating rates directly. While speed-ups of several order-of-magnitudes have been reported, top-down emulation approaches can suffer from not only worse accuracy but also a lack of energy conservation, generalization and flexibility. For example, emulators are generally tied to a specific vertical grid, and are less interpretable and configurable than modern radiation schemes which use different modules to compute the optical properties of gases, aerosols and clouds, and combine these in a radiative transfer solver. Radiative forcings with respect to individual greenhouse gases, important for climate applications, may also not be well represented by top-down emulators (we are not aware of any full-emulation paper evaluating these). The advantages of flexibility (also with regards to vertical grids) can be retained by only replacing the gas optics component with NNs (Ukkonen & Hogan, 2023; Ukkonen et al., 2020;Veerman et al., 2021). Energy conservation, meanwhile, can be ensured by predicting fluxes and computing heating rates from those using a physical equation, which has been combined with a hybrid loss function to minimize heating rate errors (Ukkonen, 2022a; Yao et al., 2023). Although emulators may yet prove useful, for instance as a way of porting code to graphics processing units (GPUs), a recent study (Ukkonen, 2022a) indicates that they suffer from similar speed-accuracy trade-offs as traditional radiation schemes: a recurrent NN approach which structurally mimics radiative transfer computations gave much better accuracy than dense networks, but also a smaller speed-up.

Fortunately, the reliable radiative transfer equations need not be sacrificed at the altar of efficiency. Algorithmic developments can, for instance, substantially reduce the number of spectral terms required for a given level of accuracy (Hogan & Matricardi, 2022). In addition, the use of code restructuring to better exploit modern CPU's probably represents an underutilized potential for many physics codes. In one case, a modern radiation scheme was made roughly 3 times faster by combining a refactoring of the solver with replacing the gas optics module with a NN version (Ukkonen et al., 2020). In another, refactoring the RRTMG radiation scheme also gave a three-fold speed-up on targeted Intel hardware (Michalakes et al., 2016). In many legacy codes, the baseline performance may be much worse (Michalakes et al., 2016). While the independent column framework used in sub-grid parameterizations enables straightforward parallelization across multiple cores, exploiting other types of parallelism offered by modern CPUs, namely SIMD (single instruction, multiple data) vectorization, or instruction-level parallelism, may be considerably more challenging. Similarly, efficient use of complex cache memory hierarchies is anything but guaranteed. For any potentially expensive physics routine that is likely called within an OpenMP loop in a NWP or climate model, it follows that knowledge of basic optimization techniques of serial code is in fact highly important, especially so as simulations are being performed at increasingly high resolution, with ever higher energy costs (Fuhrer et al., 2018).

With this in mind we describe various optimizations for CKD-based radiation codes, with a focus on ecRad (Hogan & Bozzo, 2018), a flexible and open-source radiation scheme developed at ECMWF. Our main goal was to improve the performance with ecCKD (Hogan & Matricardi, 2022), a new gas optics scheme which uses relatively few *k*-terms—only 32 for the candidate SW and LW models. While this scales down the overall cost it also reduces efficiency by shortening vectorized loops. To address this we restructure the longwave (LW) and shortwave (SW) versions of the TripleClouds and SPARTACUS solvers (Hogan et al., 2016). Our target is ECMWF's new computing platform based on AMD Zen 2 ("Rome") microarchitecture, but expressing more parallelism should help prepare ecRad for GPUs, and is generally important as hardware is evolving toward higher levels of parallelism. Our code restructuring relies on a method of batching cloudy layers that may find use in other parameterizations.

We also optimize many kernels, for example, to avoid the need for double precision in widely-used solutions to two-stream equations. However, most of the work was spent on refactoring SPARTACUS, a laborsome undertaking as the SW solver alone contained more than 1,500 lines of code when excluding subroutines. This effort should be well spent as SPARTACUS is currently the only radiation scheme that is capable of representing 3D radiative effects at a relatively low cost, having previously been 5.8 times slower than the McICA solver used in the IFS (Hogan & Bozzo, 2018). This difference is reduced by the use of ecCKD. A major goal was to eliminate the remaining gap and make SPARTACUS fast enough to be used operationally in weather and climate models. The optimization strategy relied on manually instrumenting ecRad code to estimate runtimes and floating point operations per second (FLOPS) of different code sections.

This paper is cross-disciplinary; the bulk of it concerns code optimization while its implications, scientific advances enabled by faster code, are demonstrated toward the end. Where possible, we discuss or demonstrate (using another radiation scheme) the general applicability of the optimizations. Beginning with an overview of ecRad and its relevant components (Section 2), we describe the conversion of two-stream computations to single precision (Section 3). We then describe the high-level code restructuring to increase parallelism (Section 4). Section 5 lists some general performance optimizations that were applied; more ecRad-specific optimizations are given in Appendix A. We then evaluate runtimes and performance in Section 6. Given that global simulations with SPARTACUS have not yet been published, and that the performance and spectral optimization (via ecCKD) make it fast enough for routine weather and climate simulations, some preliminary results of the impact of 3D cloud radiative effects in the IFS are presented in Section 6, followed by concluding remarks in Section 7.

2. The ECMWF Radiation Scheme "ecRad"

The ecRad radiation scheme was developed at ECMWF and has been used operationally in the IFS since 2017 (Hogan & Bozzo, 2018) and by the German Weather Service (DWD) since 2021, as well as being available for anyone to use under an open-source license. It is written in modern Fortran and is highly configurable, with the capability for the four main components (the radiative transfer solver and the calculation of the optical properties of gases, aerosols and clouds) to be changed independently of each other. Two of these components offer opportunities for a beneficial trade-off between accuracy and efficiency: the solver (Section 2.1) and the treatment of gas optics (Section 2.2).

2.1. Radiative Transfer Solvers

The solver takes as input the optical properties of the atmosphere in different spectral regions, and computes profiles of broadband fluxes from which heating rates may be computed. The main challenge is to represent sub-grid cloud structure. The McICA solver (Monte Carlo Independent Column Approximation; Pincus et al., 2003) is used operationally by ECMWF and DWD, and feeds each spectral interval of the radiative transfer calculation with a different stochastic realization of the cloud profile. The McICA implementation described by Hogan and Bozzo (2018) exactly respects the total cloud cover prescribed by the model's overlap assumptions, as well as the fraction of clouds exposed to space at each level. However, the model's assumption on sub-grid heterogeneity of cloud water content is only respected in a statistical sense, so there is a modest amount of noise in instantaneous radiative fluxes.

The TripleClouds solver (Shonk & Hogan, 2008) takes a quite different approach: each layer containing cloud is divided horizontally into three "regions," one clear and two cloudy, with the water contents of the two cloudy regions chosen to best approximate the radiative impact of the full probability distribution of cloud water assumed by the model. The model's overlap assumptions are used to pass the fluxes between adjacent layers in a way that reproduces exactly the same total cloud cover as used by McICA, but the fluxes are free from stochastic noise.

The SPARTACUS (Speedy Algorithm for Radiative Transfer through Cloud Sides) solver of Hogan et al. (2016) describes the sub-grid cloud field in the same way as TripleClouds, but terms are added to the equations to allow radiation to flow laterally between regions at a rate proportional to the assumed length of the interface between them; flows that are neglected in all operational radiation schemes worldwide. In the shortwave, this approach to representing 3D radiative transfer has been found to perform well against reference Monte Carlo radiation calculations for a wide range of cloud types (Hogan et al., 2019), capturing differences with traditional 1D radiative transfer of as much as 40 W m^{-2}. In the longwave, emission from cloud sides acts to increase the cloud radiative effect, but preliminary evaluation against Monte Carlo calculations suggests that SPARTACUS currently somewhat overestimates this 3D effect. It was reported by Hogan and Bozzo (2018) that compared to McICA, SPARTACUS makes ecRad 5.8 times slower. Thus, SPARTACUS is a good example of a parameterization that offers a more accurate representation of the real world but has until now been too expensive to deploy operationally.

2.2. The RRTMG and ecCKD Gas-Optics Scheme

The gas-optics component dictates the spectral resolution of the entire radiative transfer scheme, and scales its overall computational cost. Like the radiation schemes of many weather and climate models worldwide, ecRad by default computes the spectral absorption of gases using the Rapid Radiative Transfer Model for General Circulation Models, RRTMG (Mlawer et al., 1997), which uses a total of 140 spectral intervals in the longwave and 112 in the shortwave.

Hogan and Matricardi (2022) recently developed the ECMWF Correlated *k*-Distribution tool "ecCKD," which generates gas-optics models in the form of look-up tables that can be stored in a single configuration file. Since version 1.4, ecRad has the capability to use ecCKD gas-optics models. Hogan and Matricardi (2022) used three techniques to reduce the number of spectral intervals while retaining accuracy: the full-spectrum correlated-*k* method (Hogan, 2010; Modest & Zhang, 2002), the hypercube partition method for treating the spectral overlap of gases, and the optimization of look-up table coefficients against a set of training profiles. We use their models with 32 spectral intervals in each of the longwave and shortwave.

3. Two-Stream Kernels: Single Precision and Other Optimizations

Before applying other optimizations we would like to convert code fully to reduced arithmetic precision, as this reduces both runtime and energy consumption. The reference version of ecRad computes the two-stream solutions of reflectance and transmittance (Meador & Weaver, 1980) in double precision, as the underlying equations are numerically sensitive. This issue was noted by Cotronei and Slawig (2020), who left these computations in double precision when converting ECHAM radiation to single precision. The Meador & Weaver equations are used also in RTE + RRTMGP (Pincus et al., 2019). The kernels compute reflectances and transmittances in each layer from its optical properties τ (optical depth), ω (single-scattering albedo), and g (asymmetry factor). In the longwave, the outputs consist of reflectance R and transmittance T to diffuse incident radiation:

$$R = \gamma_2 G\{1 - \exp(-2k\tau)\} \tag{1}$$

$$T = 2kG\exp(-k\tau) \tag{2}$$

In the shortwave, three additional variables need to be computed—unscattered transmittance to direct solar radiance T_0, the reflectance to direct incoming radiation R_{dir}, and the equivalent transmittance T_{dir}:

$$T_0 = \exp(-\tau/\mu_0) \tag{3}$$

$$\begin{aligned} R_{dir} = G_{dir}\big[&(1 - k\mu_0)(\alpha_2 + k\gamma_3) - (1 + k\mu_0)(\alpha_2 + k\gamma_3)\exp(-2k\tau) \\ &- 2k\exp(-k\tau)(\gamma_4 - \alpha_2\mu_0)T^0\big] \end{aligned} \tag{4}$$

$$\begin{aligned} T_{dir} = G_{dir}\{&2k\exp(-k\tau)(\gamma_4 + \alpha_1\mu_0) \\ &- T_0\big[(1 + k\mu 0)(\alpha_1 + k\gamma_4) - (1 - k\mu_0)(\alpha_1 - k\gamma_4)\exp(-2k\tau)\big]\} \end{aligned} \tag{5}$$

where

$$k = [(\gamma_1 - \gamma_2)(\gamma_1 + \gamma_2)]^{1/2} \tag{6}$$

$$G = \big[k + \gamma_1 + (k - \gamma_1)\exp(-2k\tau)\big]^{-1} \tag{7}$$

$$G_{dir} = G\frac{\mu_0\omega}{1 - k^2\mu_0^2} \tag{8}$$

Here μ_0 is the cosine solar zenith angle and γ are exchange rate coefficients that in ecRad are computed from optical properties by using the expressions in Fu et al. (1997) (LW) and Zdunkowski et al. (1980) (SW). We find that the code can be made mostly accurate in single precision simply by using a different minimum value for the variable k in the single precision case: 10^{-4} instead of 10^{-12}. However, very rare combinations of inputs can still lead to unphysical results in the shortwave in Equations 4 and 5. This issue was solved by constraining the output variables to be positive and to not go above physical limits, by recognizing that the direct beam can either be reflected (R_{dir}), penetrate unscattered to the base of a layer (T_0), or penetrate through but be scattered on the way (T_{dir})—the rest must be absorbed. This was coded as:

$$R_{dir} \leftarrow \max[0, \min(R_{dir}, \mu_0(1 - T_0))] \tag{9}$$

$$T_{dir} \leftarrow \max[0, \min(T_{dir}, \mu_0(1 - T_0) - R_{dir})] \tag{10}$$

Here μ_0 is present because ecRad uses a convention that the direct flux is into a plane perpendicular to the sun's direction while diffuse fluxes are into a horizontal plane. After implementing the adjusted threshold and security, the mean absolute difference in SW and LW net fluxes between double and single precision computations with TripleClouds was around 0.001 Wm^{-2} for 10,000 columns saved from a high-resolution IFS simulation, and heating rate biases were close to zero.

We also removed the transmittance computation from the vectorized loop and instead call the exponential intrinsic with an array argument. For CPU's (but not necessarily vector processors or GPU's), this is faster than inlining the exponential because the operation has a large number of instructions which can lead to a pipeline stall. We found this simple change to be helpful also for RTE + RRTMGP version 1.6, speeding up the SW kernel `sw_dif_and_source` by 2.35× when using the Intel compiler and a block size of 72 (we also found it necessary to use directives to vectorize the inner loop(s)). Finally, conditionals to ensure accurate source functions when the optical depth is low were placed in a separate post-processing loop, improving performance despite some redundant computations. In the SW kernel conditionals could be removed altogether by implementing a security to avoid division by zero that is used in RTE + RRTMGP.

4. High-Level Code Restructuring to Expose More Parallelism

4.1. Motivation

In both TripleClouds and SPARTACUS, the computation of layer reflectances, transmittances and source functions take a large share of the total runtime. In the reference code, these kernels are called within a vertical loop, and contain SIMD-vectorized loops over g-points, the innermost dimension in ecRad. This is problematic for ecCKD as it results in loops that are too short (e.g., 32 iterations) to efficiently utilize modern CPU's. Similarly to a car assembly line which can produce cars at a rate that is much faster than the time taken to produce an individual car, microprocessors have a level of parallelism that comes from *instruction pipelining*. Because pipelined instructions include a wind-up and wind-down phase where microprocessor units are idling for a given number of cycles—the number of overlapped instructions, known as latency or *depth*—the throughput (number of operations per cycle) when executing N independent operations with a pipeline of depth m is given by $p = \frac{1}{1+\frac{m-1}{N}}$ (Hager & Wellein, 2010).

In the reference code, the reflectance-transmittance kernels are called inside a vertical loop and N is equal to the number of g-points. With ecCKD, $N = 32$, and to obtain a decent efficiency of for example, $p = 0.64$ results per cycle, we arrive at $m = 19$. However, complex calculations can have much longer latencies than this, with the exponential function alone having a longer latency. The computations of reflectance and transmittance using a two-stream approximation are very involved and include many high-latency operations such as floating point division. This can easily lead to the instruction stream being stalled ("pipeline bubble"). Vector or superscalar parallelism makes the situation even worse as multiple identical pipelines operating in parallel decreases the loop length of each pipe (Hager & Wellein, 2010).

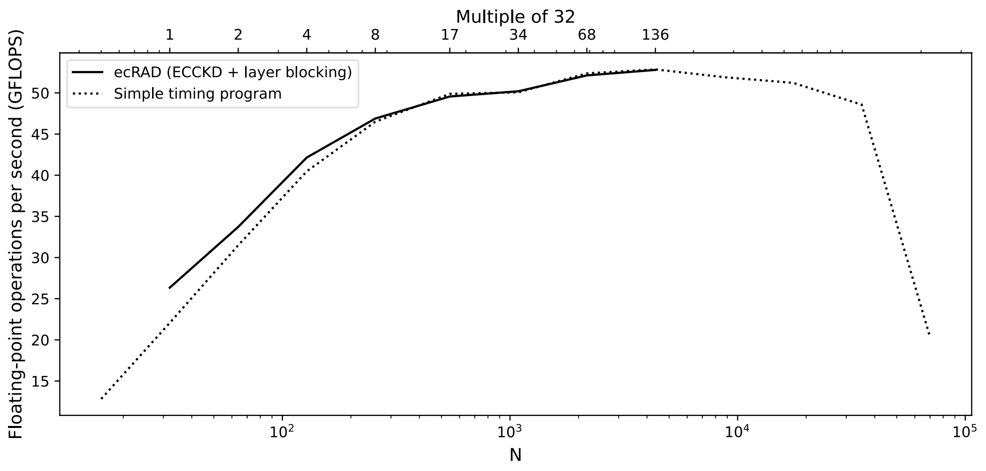

Figure 1. Serial single-precision performance of the optimized shortwave two-stream kernel (*y*-axis) versus loop length N (*x*-axis). The solid black line shows the performance when running ecRad with TripleClouds and ecCKD using a column block size of 8, and blocking also in the vertical dimension with different block sizes (top *x*-axis) to test the impact of varying N. Conveniently, the performance peaks around N corresponding to the number of *g*-points in ecCKD (32) times the number of vertical levels in the Integrated Forecast System high-resolution model (137). The dotted line was obtained using a simple timing program that calls the kernel with synthetic data in order to test a wider range of N. Platform: AMD Ryzen 9 3900, GNU Fortran 9.3 ("-O3 -march = native").

Knowing that the exponential function alone has a long latency, simply moving it outside of the long SIMD-vectorized loop with other complex arithmetic improved performance by alleviating such a pipeline stall. However, even after the separately vectorized exponential it is useful, if possible, to increase *N*. Luckily, this can be done by exploiting the lack of vertical dependencies in the underlying computations. Specifically, collapsing the vertical and *g*-point dimension together prior to the kernel calls acts to increase the length of SIMD-vectorized loops (improving vectorization and instruction-level parallelism) and also reduces overhead from procedure calls.

4.2. Batched Clear-Sky Computations

Beginning with the most trivial change, both TripleClouds and SPARTACUS compute clear-sky reflectance and transmittance for all layers (regardless of whether they contain clouds) and so the subroutine call can simply be moved outside a vertical loop and the two inner dimensions collapsed, for example, `call calc_reflectance_transmittance(ng*nlev, optical_depth(:,:,jcol), … ,)`. Here the first argument gives the length of the SIMD-vectorized dimension that is, number of *g*-points (`ng`) times number of layers, or levels (fluxes, meanwhile, are defined at `nlev+1` "half-levels"). The performance of the optimized shortwave reflectance-transmittance kernel (Section 3) as a function of the vectorized dimension *N* is shown in Figure 1. Optimal performance with ecCKD is achieved when the vertical dimension is fully collapsed with the spectral dimension, with roughly doubled performance compared to the previous code layout where the length of the vectorized loop equals `ng` = 32. It's efficient also when using other gas optics schemes, as considerably larger spectral and/or vertical dimensions can be accommodated before a performance drop-off occurs when the arrays can no longer fit in faster cache. A small trade-off is that the it requires reflectances and transmittances to be split into separate arrays for clear-sky and cloudy regions, but in practice other sections are hardly affected as flux computations depend on the presence of clouds anyway.

While the code evaluated in Figure 1 is specific to two-stream radiation schemes and arithmetically intensive, it serves as a reminder that dimension layout and code structure is a very important consideration for Earth System Models. Innermost loops over columns, as is commonplace in ESMs, may not be the optimal choice for performance given that the chunk size must typically be small (e.g., 16 columns) due to memory considerations, if parallelism is also available in the (larger) vertical dimension or other dimensions. In radiation schemes, another justification for using columns in the outermost dimension is that it avoids conditionals that are a function of column (e.g., solar zenith angle) in innermost loops where it may hinder vectorization.

```fortran
do jlev = 1,nlay ! Start at top-of-atmosphere
 nreg = nregions
 if (is_clear_sky_layer(jlev) nreg = 1

 do jreg = 1,nreg ! Loop over relevant regions (only 1 if layer is clear-sky)
  if (jreg == 1) then ! optical properties are equal to clear-sky values
   optical_depth_tot  =  optical_depth(:,jlev,jcol)
   ssa_tot = ssa(:,jlev,jcol) ! single-scattering albedo
   g_tot   =   g(:,jlev,jcol) ! asymmetry factor
  else
   do jg = 1,ng ! loop over g-points
    ! Cloudy-sky optical properties from band-wise cloud values and g-point-wise clear-sky
      values
    optical_depth_tot(jg) =  optical_depth(jg,jlev,jcol) + ...
    ...
   end do
  end if

  call calc_two_stream_gammas_sw(ng, mu0, ssa_tot, g_tot,  gamma1, ...)
  call calc_reflectance_transmittance_sw(ng, mu0, optical_depth_tot, ssa_tot, gamma1, ...,
   &  reflectance(:,jreg,jlev), transmittance(:,jreg,jlev), ... )
 end do
end do
```

$\Downarrow$

```fortran
! Reflectance-transmittance computations for clear-sky region as a separate step
! Use optimized kernel that inlines gamma computations and collapse the inner dimensions
call calc_reflectance_transmittance_sw(ng*nlay, &
  &  mu0, optical_depth(:,:,jcol), ssa(:,:,jcol), g(:,:,jcol), ! inputs &
  &  reflectance_clear, transmittance_clear, ...) ! outputs

! Start at top-of-atmosphere and find first cloudy layer, if one exists
any_clouds_below = .false.
jtop = findloc(is_clear_sky_layer(1:nlay), .false., dim=1)
if (jtop>0) any_clouds_below = .true.

do while (any_clouds_below)
 ! Find the bottom of this cloud
 jbot = ...
 nlay_cloud = jbot - jtop + 1
 allocate(optical_depth_tot_cloudy(ng,2:nreg,jtop:jbot), ssa_tot_cloudy(ng,2:nreg,jtop:
      jbot), &
    & g_tot_cloudy(ng,2:nreg,jtop:jbot))

 do jlev = jtop, jbot
  do jreg = 2, nregions ! = 3
   do jg = 1,ng
    ! Spectral cloudy-sky optical properties from band-wise cloud values and spectral
      clear-sky values
    optical_depth_tot_cloudy(jg,jreg,jlev) = ...
    ...
   end do
  end do
 end do

 call calc_reflectance_transmittance_sw(ng*2*nlay_cloud, & ! ng * cloudy regions * cloudy
      layers
  &  mu0, optical_depth_tot_cloudy, ssa_tot_cloudy, g_tot_cloudy, & ! inputs
  &  reflectance(:,:,jtop:jbot), transmittance(:,:,jtop:jbot), ...) ! outputs

 deallocate(optical_depth_tot_cloudy, ssa_tot_cloudy, g_tot_cloudy)

 ! Does another cloudy layer exist? If not, set logical to false to exit "while"
 if (jbot== nlay) any_clouds_below=.false. ! surface reached

 if (any(.not. is_clear_sky_layer(jbot+1:nlay))) then
  ! find the new cloud top
  jtop = ...
 else
  any_clouds_below=.false.
 end if
end do
```

Figure 2. Refactoring of TripleClouds-SW. In addition to optimizing and fusing kernels, in the new code (bottom) the reflectance-transmittance computations are performed in a batched manner for multiple layers by collapsing the spectral and vertical dimensions.

4.3. Batched Cloudy Computations

The lack of loop dependencies in the vertical dimension can likewise be exploited in the more demanding reflectance-transmittance computations for cloudy layers and regions, but this requires batching together the two cloudy regions and/or adjacent cloudy layers. The best way to do this depends on the particular solver.

4.3.1. TripleClouds

In shortwave TripleClouds, we collapse the *g*-point, region and vertical dimensions by grouping together adjacent cloudy layers. This was implemented with a do while loop which checks if any cloudy layers still exists and finds the top and bottom of this extended cloudy layer, as illustrated in Figure 2. The new code leads to a vectorized dimension of $\texttt{ng} \times 2 \times \texttt{nlay\_cloud}$ in the cloudy reflectance-transmittance computations. In longwave TripleClouds we decided to batch the reflectance-transmittance computations only over *g*-points and the two cloudy regions, but not layers, as this was slightly faster on the tested platform. To achieve better performance on platforms with longer vector lengths it would likely be worth the increase in memory footprint to batch over the vertical dimension as well.

4.3.2. SPARTACUS

SPARTACUS represents cloud 3-D radiative effects by adding extra terms to the two-stream equations. The coupled system of equations can be solved by a method based on the matrix exponential. Despite using an optimal scaling and squaring algorithm for this problem, the kernel is relatively expensive: in the reference code `expm` accounts for nearly 50% of the total runtime. These computations are performed for each "3D" *g*-point in each cloudy layer; 3D effects being ignored for *g*-points which have very large optical depths that exceed a threshold. Because the individual matrices for which the matrix exponential is computed are small (9-by-9) in the shortwave they are placed non-contiguously in memory and the *g*-point dimension is vectorized instead. To vectorize over "3D" *g*-points, it is assumed that prior to the solver the *g*-points have been reordered in approximate order of gas optical depth which in practice is implemented using a hard-coded mapping. Clear-sky optical depths are then searched for the cut-off index `ng3D` used in `expm`, which is dominated by matrix-matrix multiplications implemented as $\texttt{C(1:ng3D,j1,j2) = C(1:ng3D,j1,j2) + A(1:ng3D,j1,j3)} \times \texttt{B(1:ng3D,j3,j2)}$. Optimization of the shortwave kernel `expm_sw` is described in Appendix A.

Efficiency can again be improved by batching adjacent cloudy layers within a do while loop. Recognizing that in the shortwave, `ng3D` is typically close to `ng`, 3D computations can be performed for all *g*-points without much redundancy (capping the optical depths to the threshold value), and collapsing the spectral and vertical dimensions. This results in a vectorized dimension of $\texttt{ng} \times \texttt{nlay}_{\text{cloud-depth}}$ instead of $\texttt{ng3D} \approx \texttt{ng}$, and in addition eliminates errors associated with assuming a constant reordering. Unlike TripleClouds, the computations in SPARTACUS involve many large intermediate arrays, and to improve the use of cache memory it's in this case useful to limit the number of batched cloudy layers. We set the maximum batch size with a simple expression that depends on `ng`, working precision, and a constant tuned for optimal performance on the AMD platform (resulting in six layers when using 32 *g*-points and single precision); this could be further tuned for specific hardware.

Finally, after reflectances and transmittances have been determined, the solver works its way up from the surface to the top-of-atmosphere computing the total albedos (the albedo of the entire atmosphere below a layer). In the shortwave, this includes the computation of entrapment (Hogan et al., 2019) where the rate of exchange between the subregions in a given layer and the subregions in the layer above is once again solved using the matrix-exponential method. The simpler structure of these matrices enables using a faster method described in the appendix of Hogan et al. (2018). Nonetheless, these computations represent a small hotspot. While the loop-carried dependencies prevent batching across the vertical dimension as for `expm`, it is possible to batch the `fast_expm` computations across the three subregions times two (being performed for both diffuse and direct albedo), increasing the vectorized dimension sixfold.

In the longwave, the fraction of g-points which have optical depths small enough for 3D effects to matter is typically much lower than in the shortwave, and doing them for all g-points would result in a great deal of redundancy. Therefore, the code was restructured to collect all the "3D" g-points from adjacent cloudy layers, where `ng3D` varies by layer, into larger arrays with the inner dimension $\texttt{ng3D}_{\text{tot}}$. This increases code complexity and introduces overhead but is worth it as the time spent in `expm` was more than halved (when using ecCKD and optimized kernel) due to avoiding very inefficient calls with small loop lengths. This change made SPARTACUS-LW faster by roughly a third.

4.3.3. Application to Other Schemes

We have described the cloudy batching method for ecRad in the hope that the principle could be applied to expose more parallelism in other parameterizations that perform demanding computations specific to cloudy layers (e.g., cloud microphysics schemes). In this case, the lack of another dimension (here the spectral) to collapse the vertical with would result in much shorter loop lengths. It may therefore be useful to modify the method; for instance to gather all the cloudy levels in a column (not only adjacent layers) into a contiguous input array before performing expensive computations.

5. Other Optimizations

To further improve performance we made use of many general optimization techniques for serial code that may be useful for other parameterizations. We list these below while referring the reader to Appendix A for more ecRad-specific optimizations.

Loop unrolling, loop fusion and function inlining. Short loops related to regions were completely unrolled in many places, and many loops were also fused. This was often made possible by manually inlining functions, which in itself can improve performance by reducing overhead and allowing the compiler to use registers and employ optimizations that require a larger view of the code (Hager & Wellein, 2010).

Declaring `ng` at compile time. The fastest-varying dimension in ecRad is over g-points and in many code sections it cannot be collapsed with the vertical dimension. Simply declaring $\texttt{ng}_{\text{SW}}$ and $\texttt{ng}_{\text{LW}}$ at compile time reduces ecRad's runtime with ecCKD by up to 25% (Section 6) by allowing the compiler to optimize many such short loops in the solvers, aerosol optics and gas optics. This was implemented using a preprocessing directive `#ifdef ng_sw` which sets the leading dimension to a parameter if it is passed to the compiler, and to a procedure argument `ng_sw_in` if it is not.

Removing conditionals. Conditional branches to prevent division by zero, for example, in sections where optical properties from gases, clouds and aerosols are combined within a spectral loop, were replaced with the use of max(*value*, *some number*) in the denominator by recognizing that if the denominator was zero the numerator was also zero. In the LW two-stream kernel moving a necessary conditional to a separate loop also improved performance by vectorizing the more compute-intensive parts.

Merged broadband flux computations. The last step in the solver is to compute broadband fluxes by summing the fluxes defined at g-points and three regions. In the shortwave, this reduction over two dimensions is performed for three variables: upwelling, downwelling, and direct downwelling flux. By doing all three sums in a single loop over g-points with the *SIMD reduction* clause in OpenMP, and manually unrolling the sum over regions, the arithmetic intensity can be greatly improved compared to having separate calls to the *sum* intrinsic function:

```
sums_up = 0.0; sums_dn = 0.0; sums_dn_dir = 0.0
!$omp simd reduction(+:sums_up, sums_dn, sums_dn_dir)
do jg = 1, ng_sw
   sums_up = sums_up + flux_up(jg,1) + flux_up(jg,2) + flux_up(jg,3)
   sums_dn = …
end do
flux%sw_up(jcol,jlev+1) = sums_up
```

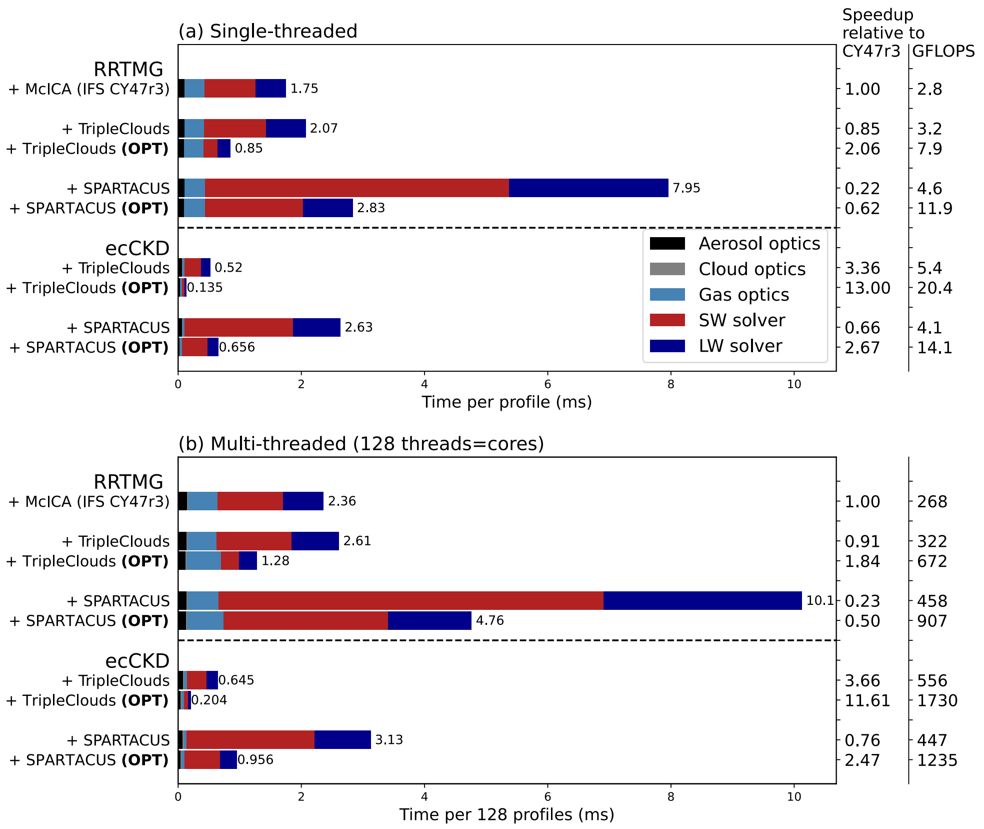

Figure 3. (a) Time per profile (x-axis) for different configurations of ecRad (y-axis) run single-threaded and (b) time per 128 profiles when using 128 CPU cores (i.e., time per profile in equivalent single-threaded time), with colors indicating different components. The results are grouped first by the choice of gas optics (RRTMG or ecCKD) as this determines the number of g-points. Then, the results are grouped by solver, and finally (for TripleClouds and SPARTACUS only) by different versions of code, where the runtime profile of the optimized code (OPT) follows that of the reference. To the right, speedup w.r.t. the configuration of ecRad in Integrated Forecast System cy47r3 (RRTMG + McICA) followed by FLOPS (obtained using the General Purpose Timing Library) is shown. In (b) the component runtimes are means of per-thread values, normalized to add up to the total runtime. Platform: AMD EPYC 7H12, GNU Fortran compiler version 9.3 ("-O3 -march = native").

The fused reduction using OpenMP is also beneficial (but less so) for spectral-innermost radiation schemes that do not use TripleClouds' partition of regions. When evaluated on the same platform as in Figure 1, it sped up the SW solver of RTE + RRTMGP-NN (Ukkonen et al., 2020) by 4%, and of TripleClouds by 18%. We also eliminated redundant summation over cloudy regions in cloud-free layers, yielding a combined speed-up of 30%.

Avoiding temporary arrays. In many sections, one or more temporary arrays were removed by using the output array(s) of a subroutine for intermediate computations and/or by reusing temporary/local arrays. Code clarity was retained by the use of Fortran's *associate* construct.

6. Timing Results

We evaluate performance by running ecRad on ECMWF's new AMD-based supercomputer using 10,000 input columns randomly sampled from a global snapshot of a high-resolution IFS simulation with 137 vertical levels. As in the IFS, we configure ecRad so that longwave scattering by clouds (but not aerosols) is represented. Figure 3 shows the runtimes with a breakdown into components, as well as the overall single-precision floating-point performance. To facilitate comparison with other studies both single-threaded and multi-threaded timings were obtained with the fastest of 5 runs shown in each case. The dynamically scheduled OpenMP parallelization was over blocks of columns (block size was set to 8) in an outer loop, in which the ecRad derived type arguments, and not their array components, are blocked in order to avoid inefficient striding over all columns (unlike ecRad's internal variables, its input/outputs use columns innermost). This reflects IFS use, except that the offline setup

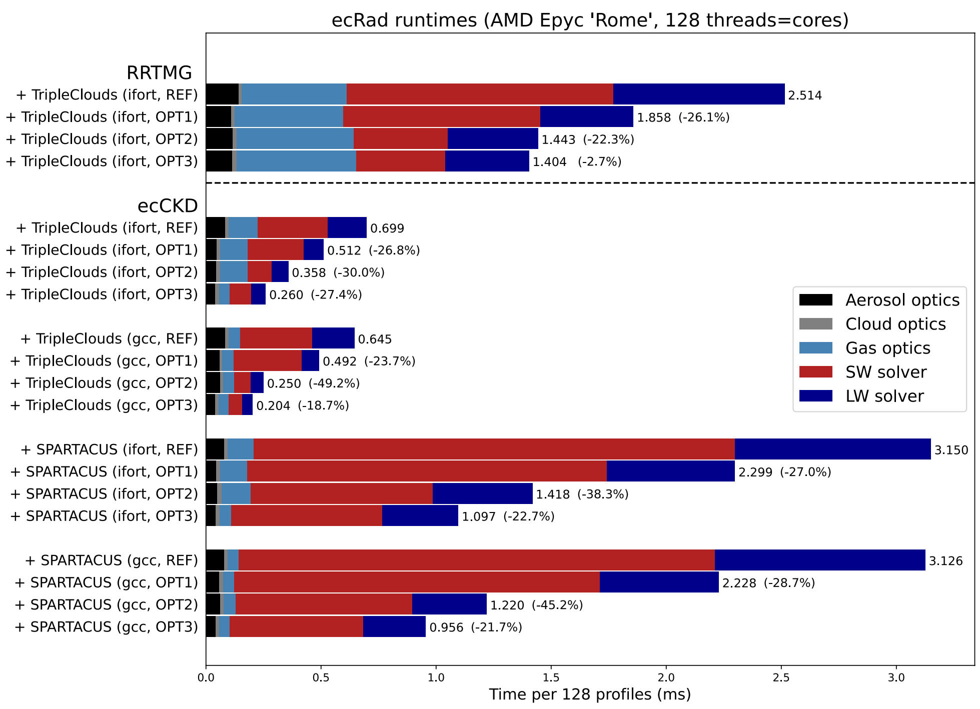

Figure 4. As in Figure 3 (b), but for increasing levels of code refactoring and both the GNU Fortran (labeled "gcc") and Intel Fortran compiler (with compiler options "-O2 -march = avx2 -align array64byte -fast-transcedentals -finline-functions …" reflecting Integrated Forecast System use) included in Intel OneAPI version 2021.4. The change in runtime relative to the previous level of optimization is shown in brackets. OPT1 = all changes except using the original reflectance-transmittance and matrix exponential kernels, and without declaring `ng` at compile time. OPT2 = OPT1 + optimized main kernels. OPT3 = OPT2 + declaring `ng` at compile time (full optimizations, corresponding to "OPT" in Figure 3).

does not include preparation of derived types and interpolation to the coarser grid. In the OpenMP runs the workload was increased by a factor of 40.

The optimizations give a 3–4× speed-up in the total runtime of ecRad configured with ecCKD and either TripleClouds and SPARTACUS. Optimized TripleClouds with ecCKD is very fast: a profile with 137 levels takes only 0.0135 milliseconds to compute on a single core. Using the full 128-core node decreases the runtime by 85× due to imperfect scaling (presumably caused by cores sharing L3 cache memory), with the corresponding throughput at 629 profiles ms^{-1}. In a multi-threaded setting, optimized TripleClouds with ecCKD is 11.6× faster than the operational IFS radiation (reference ecRad using McICA and RRTMG), achieved mainly by the reduction in spectral resolution (64 vs. 252 g-points in total) combined with a much higher floating point performance (1,730 vs. 268 GFLOPS), as opposed to fundamental differences between the solvers (their reference versions have similar runtimes and FLOPS). As for SPARTACUS, we find that its optimized version with ecCKD runs 2.5× faster than operational IFS radiation, and 10.6× faster than reference SPARTACUS with RRTMG, making cloud 3D effects fully affordable to compute in large-scale dynamical models. Performance with other gas optics schemes is also improved: ecRad with RRTMG is sped up 2×.

The speed-ups are a result of a large number of changes. To assess their relative importance our version of offline ecRad can be compiled with three levels of increased refactoring. The runtimes using different versions of the code and two different compilers (including Intel's compiler, which is used for the operational forecast model at ECMWF) are shown in Figure 4. It can be seen that both high-level refactoring and kernel-level optimizations are important, but the latter are decisive in achieving high performance and getting the full benefit of layer batching, as switching to the new reflectance-transmittance and `expm` kernels (the main hotspots) gives the largest percentage reduction in runtime relative to the previous level of code optimization. Finally, making `ng` a compile time constant speeds up radiation computations with 32-term ecCKD models by a further 19%–27%, having a larger impact when using the Intel compiler.

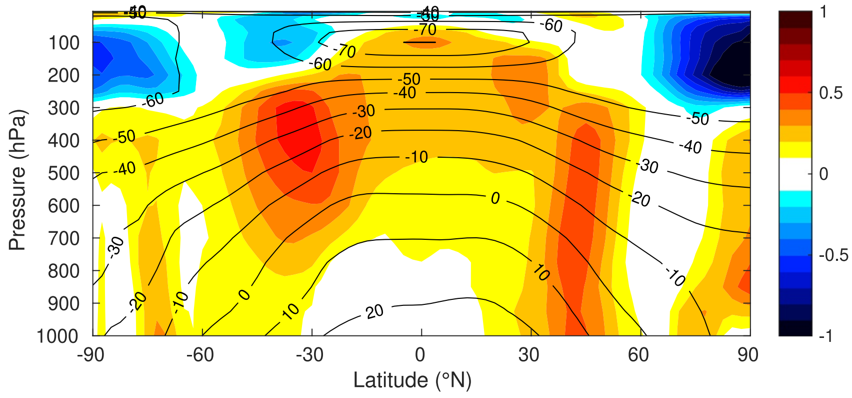

Figure 5. Height-latitude cross section of the zonal mean of the temperature difference between year-long Integrated Forecast System simulations using SPARTACUS and TripleClouds (where the former includes cloud 3D radiative effects).

7. Preliminary IFS Results With SPARTACUS

Having demonstrated the optimizations that make the SPARTACUS solver computationally affordable, we now briefly describe the impact of cloud 3D radiative effects in IFS cy47R3 by comparing simulations using SPARTACUS and TripleClouds, where the latter is a solver that is equivalent to SPARTACUS but does not compute 3D effects. First, to estimate the impact on model climate, eight 13-month long (first month is spin-up) coupled atmosphere-ocean simulations using a horizontal grid spacing of around 60 km ($T_{Co}199$) were performed. These simulations are similar to the seasonal forecasts performed operationally at ECMWF, and are long enough to capture fast atmospheric and land-surface processes that respond to changes in the radiation scheme, but short enough that the response is not significantly affected by the longer-term changes to ocean circulation. We note that while 3D effects have an overall surface warming effect on larger scales, they include several processes such as shortwave cloud side interception whose cooling effect can dominate at low solar zenith angles; this could be seen if looking at instantaneous and local 3D effects as opposed to long-term averages (Schäfer, 2017), which is our focus here. The other 3D effects that are represented by SPARTACUS include leakage of shortwave and longwave radiation from cloud sides, longwave emission through cloud sides, and shortwave "entrapment" (Hogan et al., 2019), all of which act to increase downwelling flux below the cloud.

Figure 5 shows a latitude-pressure cross-section of zonal mean temperature differences between the SPARTACUS and TripleClouds runs. In year-long simulations, 3D effects warm almost the entire troposphere by up to 0.5 K, the warming being strongest at mid-latitudes, while impacts are neutral below 700 hPa near the equator. The impact on top-of-atmosphere (TOA) and surface net shortwave and longwave fluxes is shown in Figure 6. The simulations using SPARTACUS show a net increase in both shortwave and longwave radiation at TOA relative to TripleClouds (a warming of the climate system), $+0.59 \pm 0.41$ W m^{-2} in the shortwave and $+0.46 \pm 0.36$ W m^{-2} in the longwave. These numbers include the 95% confidence interval computed assuming each member of the 8-member ensemble is an independent sample. The longwave 3D effects are even stronger at the surface (where unlike in the shortwave, the longwave 3D effects always have a direct warming effect), causing a net increase of 0.96 ± 0.31 W m^{-2} in the year-long simulations.

Hogan et al. (2019) rigorously evaluated the shortwave component of SPARTACUS against Monte Carlo calculations across a diverse range of cloud scenes, which led to significant improvements being made, but the same has not yet been done with the longwave component. Therefore, the results in this section should be regarded as preliminary. Schäfer et al. (2016) did compare longwave SPARTACUS to Monte Carlo for a single cumulus scene, and found that it tended to overestimate the 3D effect somewhat, but that good agreement was achieved by reducing cloud edge length (which in ecRad is parameterized following Fielding et al., 2020) by a factor of 0.69. They attributed this to cloud clustering, that is, the tendency of clouds to lie closer to one another than the random distribution assumed implicitly by SPARTACUS. While this factor is unlikely to be exactly appropriate for all cloud types globally, it can be used to provide a rough estimate of the impact of uncertainties in both the longwave component of SPARTACUS, and the representation of cloud geometry. We carried out another set of eight year-long experiments where we reduced the cloud edge length by a factor of 0.69 (in both the shortwave

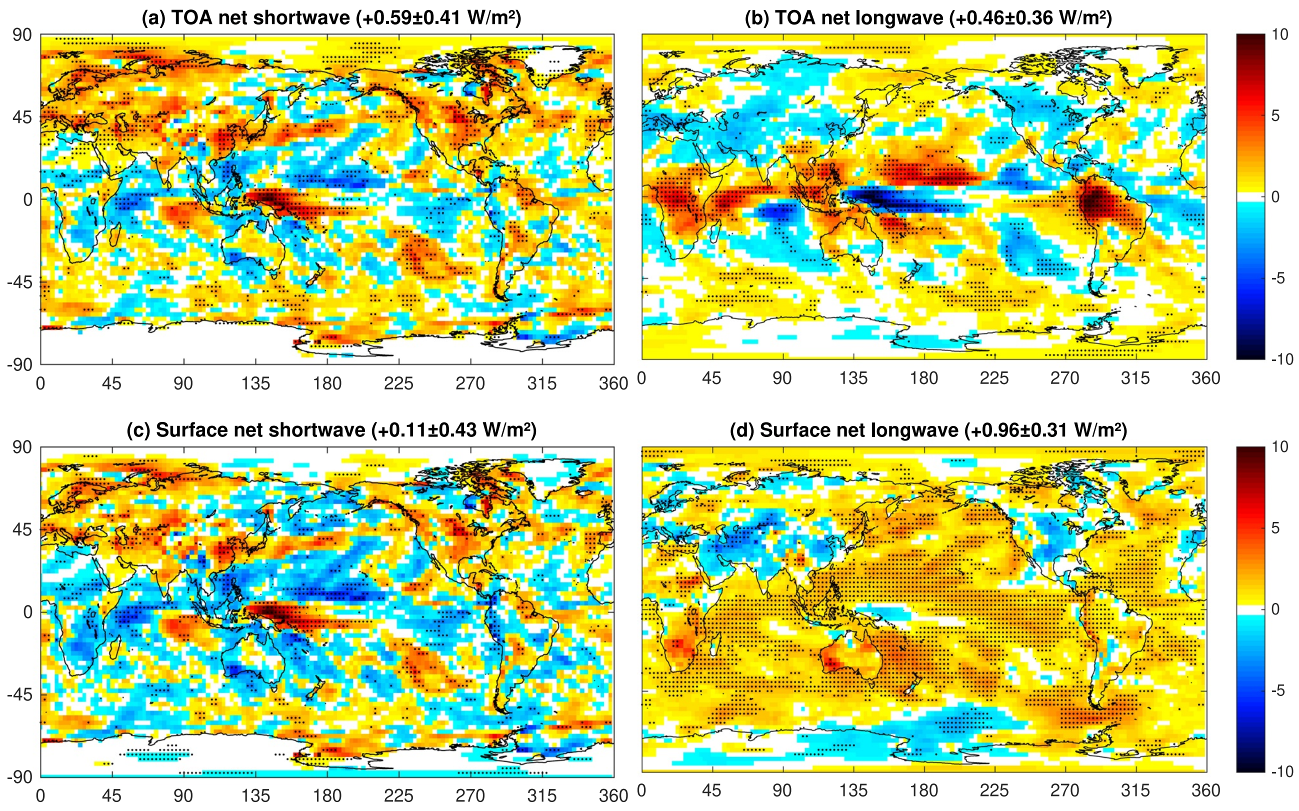
(a) TOA net shortwave (+0.59±0.41 W/m²)
(b) TOA net longwave (+0.46±0.36 W/m²)
(c) Surface net shortwave (+0.11±0.43 W/m²)
(d) Surface net longwave (+0.96±0.31 W/m²)
90
45
0
-45
-90
0 45 90 135 180 225 270 315 360
10
5
0
-5
-10

Figure 6. (a and b) Top-of-atmosphere and surface (c and d) net shortwave (a and c) and (b and d) longwave flux in year-long SPARTACUS simulations relative to TripleClouds, showing a warming of the climate system from cloud 3D radiative effects. Numbers in subcaptions give the global mean forcing with 95% confidence interval, and dotted locations are where the changes are statistically significant (signal exceeds the 95% confidence interval).

and longwave). This decreased the (SPARTACUS − TripleClouds) change in longwave cloud radiative forcing (CRE) at TOA from $+0.92 \pm 0.14$ W m^{-2} in the reference runs to $+0.71 \pm 0.12$ W m^{-2} (not shown).

We stress that these simulations are too short to capture the ocean response (acting to underestimate the impact of 3D effects on climate) and that SPARTACUS is under development to improve realism in the longwave (where it appears to currently overestimate 3D effects). But interestingly, a visual comparison with Figure 2 of Tian et al. (2013), depicting CMIP5 tropospheric temperature biases against the MERRA reanalysis and a satellite infrared product, suggests a reasonable match between the SPARTACUS warming pattern and CMIP5 cold biases. Comparing our IFS simulations to ERA5, some existing mid-latitude cold biases were indeed reduced, but SPARTACUS also introduced a warm bias in low latitudes between 200 and 700 hPa, and exacerbated existing IFS stratospheric cold biases near the poles (not shown), where 3D effects have a cooling effect that reaches 1 K over the North Pole. Because operational models are tuned to improve the model climate and contain numerous compensating errors, tuning or revision of other model components is likely required to offset the temperature changes caused by SPARTACUS.

Finally, we consider the impact on forecast skill using a suite of high-resolution (TCo1279; roughly 9 km horizontal grid spacing) 10-day simulations initialized at consecutive days between 1. June and 31. August 2021 (a total of 92 runs using both TripleClouds and SPARTACUS). Given that we did not perform any model tuning, it's perhaps not surprising that the SPARTACUS runs exhibit higher root-mean-square-error (RMSE) in temperature aloft due to increased bias, with significant skill degradation in the low latitudes between 100 and 900 hPa (due to warming), and in the northern hemisphere between 10 and 100 hPa (due to cooling). This is not shown, instead we focus on the areas where we find improvement. Most notably, RMSE of 2-m temperature is reduced by up to 10% in the tropics (Figure 7). The decrease in RMSE over tropical land was partially due to a reduced cold bias. But encouragingly, the standard deviation of 2-m temperature was also significantly reduced in the tropics, by nearly 1% on average between 20°N and 20°S (not shown). Random error of low cloud cover in the tropics was also slightly improved.

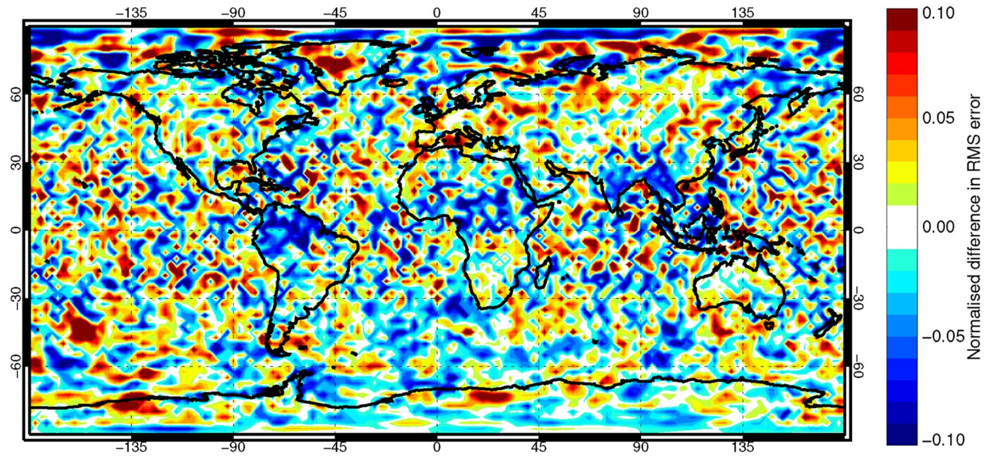

Figure 7. Normalized difference in root-mean-square-error in the 7-day forecast of 2-m temperature between high-resolution simulations using SPARTACUS and TripleClouds. The plot shows the average impact on forecast skill across a suite of TCo1279 Integrated Forecast System simulations in June-July-August 2019 (82 samples). Blue-colored areas are those where SPARTACUS decreases RMSE (up to 10% over tropical land).

8. Conclusions

In this work the ecRad radiation scheme (https://github.com/ecmwf-ifs/ecrad) has been optimized by using both kernel-level optimizations and higher-level code restructuring. Some of our optimizations—for example, porting two-stream kernels to single precision—are directly applicable to other radiation codes. The code restructuring addresses recent developments in gas optics schemes, namely the ecCKD tool, which allows the spectral dimension to be reduced considerably (to e.g., 32 g-points in the LW and SW) while retaining accuracy. While speeding up all ecRad solvers, it also decreases floating-point performance due to shortening vectorized loops over *g*-points. We therefore restructured the TripleClouds and SPARTACUS solvers to collapse the spectral and vertical dimensions where possible. We also performed many lower-level optimizations, for instance to improve the efficiency of matrix computations in SPARTACUS, a solver that can compute cloud 3D radiative effects at a relatively low cost. In an effort to make it truly affordable for operational use, we ended up carrying out a thorough performance refactoring of the entire SPARTACUS code. Our full optimizations speed up ecRad configured with ecCKD and either TripleClouds or SPARTACUS by 3-4×, and the optimized code is also much faster when using older gas optics schemes with more *g*-points.

While targeting ECMWF's new supercomputer equipped with AMD Zen 2 CPUs, exposing more parallelism via code restructuring should be useful for any future code porting on GPU, and benefit CPU's with longer vector lengths (via AVX-512 instructions) even more. It may be applicable to other correlated-*k* radiation codes, or possibly even other physics parameterizations which include demanding computations conditional to the presence of clouds. The spectral-innermost memory layout of ecRad, when combined with code restructuring to group together cloudy layers and collapsing with the spectral dimension, is likely ideal for performance for 1D radiation schemes as it allows for sufficiently long vectorized loops (even for spectrally reduced gas optics) to achieve high performance. A memory layout with columns innermost would not allow any compute-intensive computations that are specific to cloudy layers to be batched in a similar way, and the column batch size may have to be kept small due to memory constraints, reducing SIMD and instruction-level parallelism.

Combining optimized TripleClouds with ecCKD, we obtain a speed-up of 12× relative to the operational radiation scheme in IFS cy47r3, which is based on McICA and RRTMG. This may have implications for emulation studies, which attempt to replace physical schemes with a cheaper NN emulator: considering that a low-complexity recurrent NN (which, unlike a faster dense NN, could produce both fluxes and heating rates accurately) was only 4× faster than a shortwave radiation scheme which uses 7× more *g*-points than ecCKD (Ukkonen, 2022a), one may question the value of emulation—at least for 1D radiation schemes seeking a speed-up on CPUs. Future studies should strive to compare NNs to state-of-the-art radiation schemes, as older codes may be orders-of-magnitudes slower. (As computational results depend on the hardware and software platform, such details should also be mentioned.) For example, Lagerquist et al. (2021) claimed a speed-up of 10^4 using emulators, but reported their RRTM radiation scheme taking 1,200 ms per profile, whereas Hogan and Bozzo (2018) reported 7.5 ms per profile for the ecRad implementation of RRTMG (160x faster despite using more vertical levels; 137 vs. 73). The latter was evaluated single-threaded on the older Cray-based HPC at ECMWF. On the new AMD-based HPC our optimized TripleClouds with ecCKD takes only 0.134 ms per profile single-threaded; a 10^4 speed-up over the 1,200 ms reported in Lagerquist et al. (2021) which would imply similar speed as the emulator.

Finally, we find that optimized SPARTACUS coupled with ecCKD is 2.5× faster than the operational IFS radiation. To our knowledge, cloud 3D radiative effects have until now been neglected in all weather and climate models due to computational reasons, so this represents a major development. In year-long coupled IFS simulations, SPARTACUS significantly warms the troposphere compared to its fully-1D counterpart (TripleClouds), and these effects are likely to be more pronounced in longer climate simulations, which we leave for future studies to explore. We also performed high-resolution simulations and find that SPARTACUS improves medium-range forecasts of 2-m temperature and low cloud cover in the tropics. SPARTACUS is still under development to improve some physical assumptions made in the longwave, and we also foresee other opportunities to further increase realism, such as using high-resolution model cloud fields to determine inputs related to cloud sub-grid variability when running radiation on a coarser grid, as is currently done in the IFS.

Appendix A: Kernel-Level Optimizations

A1. SPARTACUS Matrix Operations

Loop unrolling is a common optimization strategy that compilers can in some cases perform automatically. However, more involved code patterns may prevent the compiler from doing this, or it may not know it is advantageous if loop bounds are unknown at compile time. SPARTACUS uses a matrix exponential solver based on a single precision variant of an optimal scaling and squaring algorithm utilizing Padé approximants (Higham, 2005). The scaling and squaring method involves performing many matrix-matrix multiplications. Because the matrices operated by SPARTACUS are very small, ($\texttt{nreg} \times 3$, $\texttt{nreg} \times 3$) = (9, 9) in the shortwave and ($\texttt{nreg} \times 2$, $\texttt{nreg} \times 2$) = (6, 6) in the longwave, for performance reasons the matrix-exponential kernel `expm` stores them in the two outer dimensions of 3D arrays and the fastest-varying spectral dimension is vectorized instead. We found that manually unrolling the innermost of the matrix multiplication loops improved performance on the tested compilers. Redundant computations in `expm` were also identified and removed: in the shortwave (only), many of the matrix-matrix multiplications can exploit not only the sparsity but also some repeated elements in the input matrices, which result in the output matrices also having repeated elements. Given this and the different matrix dimensions, separate LW and SW versions were written for `expm`. The refactoring of the shortwave matrix multiplication kernel is illustrated in Figure A1.

Similar optimizations were also employed in the many other matrix operations performed by SPARTACUS, such as matrix-vector multiplication, and solving linear systems of equations for a matrix or vector using LU decomposition. For most of these, separate longwave and shortwave kernels were made to allow declaring the inner dimension ($\texttt{ng}_{\mathrm{SW}}$ or $\texttt{ng}_{\mathrm{LW}}$) at compile time, even if other dimensions were identical.

The other main optimization for `expm` was in the last step of the algorithm, where the matrices for different g-points are individually squared. This section has poor performance because the number of squarings (stored in the N-sized integer array `expo`) varies by g-point, resulting in many temporary copies of small arrays and lack of vectorization. Efficiency was improved by first squaring all the matrices by the minimum `expo`, ensuring vectorization. In the shortwave, performance was also increased (at the cost of code complexity) by squaring groups of matrices, based on array indexing of memory-contiguous matrices that still need to be squared after the first step.

```
function mat_x_mat(ng,ng3D,m,A,B,i_matrix_pattern) result(C)
  integer,    intent(in)                          :: ng, ng3D, m
  integer,    intent(in), optional                :: i_matrix_pattern
  real(jprb), intent(in), dimension(:,:,:)        :: A, B ! (ng,m,m)
  real(jprb),             dimension(ng3D,m,m)     :: C
  ! Treat A and B each as n m-by-m square matrices (with the n dimension
  ! varying fastest) and perform matrix multiplications on all n matrix pairs
  C = 0.0_jprb ! Array-wise assignment
  mblock = m/3
  m2block = 2*mblock
  if (i_actual_matrix_pattern == IMatrixPatternShortwave) then
    ! Matrix has a sparsity pattern
    !     (a b c)
    !     (d e f)
    !     (0 0 g)
    ! Do the top-left (a, b, d, e)
    do j2 = 1,m2block ! 1,6
      do j1 = 1,m2block ! 1,6
        do j3 = 1,m2block ! 1,6
          C(1:ng3D,j1,j2) = C(1:ng3D,j1,j2) + A(1:ng3D,j1,j3)*B(1:ng3D,j3,j2)
        end do
      end do
    end do
    do j2 = m2block+1,m ! 7,9
      ! Do the top-right (c, f)
      do j1 = 1,m2block ! 1,6
        do j3 = 1,m ! 1,9
          C(1:ng3D,j1,j2) = C(1:ng3D,j1,j2) + A(1:ng3D,j1,j3)*B(1:ng3D,j3,j2)
        end do
      end do
      ! Do the bottom-right (g)
      do j1 = m2block+1,m ! 7,9
        do j3 = m2block+1,m ! 7,9
          C(1:ng3D,j1,j2) = C(1:ng3D,j1,j2) + A(1:ng3D,j1,j3)*B(1:ng3D,j3,j2)
        end do
      end do
    end do
  else
  ...
```

$$\Downarrow$$

```
pure subroutine mat_x_mat_sw_repeats(ng_sw_in, nlev_b, A, B, C)
  integer,    intent(in)                                :: ng_sw_in, nlev_b
  real(jprb), intent(in), dimension(ng_sw*nlev_b,9,9) :: A, B
  real(jprb), intent(out),dimension(ng_sw*nlev_b,9,9) :: C
  integer    :: j1, j2, j22
  !dir$ assume_aligned A:64,B:64,C:64
  ! Input matrices have pattern:
  !   (a          b           c)
  !   (d=-b       e=-a        f)
  !   (0          0           g), where each element is a 3-by-3 matrix
  ! As a result, output matrices have pattern:
  !   (a          b           c)
  !   (d=b        e=a         f)
  !   (0          0           g)
  do j2 = 1,3
    j22 = j2 + 6
    do j1 = 1,6
      ! Do the top-left (a & b)
      ! Unroll innermost matmul loop: more work for each iteration of SIMD loop
      C(:,j1,j2) = A(:,j1,1)*B(:,1,j2) + A(:,j1,2)*B(:,2,j2) + A(:,j1,3)*B(:,3,j2) &
      &          + A(:,j1,4)*B(:,4,j2) + A(:,j1,5)*B(:,5,j2) + A(:,j1,6)*B(:,6,j2)
      ! Do the top-right (c & f)
      C(:,j1,j22) = A(:,j1,1)*B(:,1,j22) + A(:,j1,2)*B(:,2,j22) + A(:,j1,3)*B(:,3,j22) &
      &      + A(:,j1,4)*B(:,4,j22) + A(:,j1,5)*B(:,5,j22) + A(:,j1,6)*B(:,6,j22) &
      &      + A(:,j1,7)*B(:,7,j22) + A(:,j1,8)*B(:,8,j22) + A(:,j1,9)*B(:,9,j22)
    end do
    do j1 = 7,9  ! Do the bottom-right (g)
      C(:,j1,j22) = A(:,j1,7)*B(:,7,j22) + A(:,j1,8)*B(:,8,j22) + A(:,j1,9)*B(:,9,j22)
    end do
  end do
  C(:,1:3,4:6) = C(:,4:6,1:3) ! b = d
  C(:,4:6,4:6) = C(:,1:3,1:3) ! e = a
  C(:,7:9,1:6) = 0.0_jprb     ! Lower left corner
```

Figure A1. Reference (top) and optimized (bottom) versions of the matrix-matrix multiplication kernel used in the shortwave matrix exponential computations. The latter unrolls loops and reduces work by exploiting that some matrix elements are repeated. For this performance-critical code, further speedup was gained by data alignment. The Intel compiler reported aligned data access only after declaring `ng_sw` at compile-time.

A2. Longwave Derivatives

The final step in the longwave solvers is the computation of longwave derivatives, the rate of change of layer broadband upwelling longwave fluxes with respect to surface broadband upwelling flux, which is used for approximate radiation updates in every model column at every model time step (Hogan & Bozzo, 2015). This kernel was relatively expensive for TripleClouds, as it consists of doing `ng` multiplications of very small matrices and vectors (`m` = `nreg`), followed by a multiplication with transmittance (`ng`,`nreg`) at each g-point, and finally a sum over `ng` and `nreg`, at each level. In the expected case of `nreg=3`, the matrix-vector computations, multiplication with transmittance and sum over `nreg` and `ng` were all combined in a single vectorized loop over *g*-points by inlining the matrix-vector computation and unrolling the three regions (Figure A2). When also making `ng` a compile-time constant, the kernel was sped up by a factor of 5–7, decreasing its share of the total runtime from almost a fifth to only a few percent. A similar optimization was done for SPARTACUS where transmittances are 3-D arrays.

```
! Initialize the derivatives at the surface; the surface is treated as a
! single clear-sky layer so we only need to put values in region 1.
lw_deriv_g_region = 0.0_jprb
lw_deriv_g_region(:,1) = flux_up_surf / sum(flux_up_surf)
lw_deriv(icol, nlev+1) = 1.0_jprb

! Move up through the atmosphere computing the derivatives at each half-level
do jlev = nlev,1,-1
  ! Compute effect of overlap at half-level jlev+1, yielding derivatives just above
  ! that half-level (matrix-vector multiply)
  lw_deriv_g_region = singlemat_x_vec(ng,ng,nreg,u_matrix(:,:,jlev+1),lw_deriv_g_region)

  ! Compute effect of transmittance of layer jlev, yielding
  ! derivatives just below the half-level above (jlev)
  lw_deriv_g_region = transmittance(:,:,jlev) * lw_deriv_g_region

  lw_deriv(icol, jlev) = sum(lw_deriv_g_region)
end do
```

$$\Downarrow$$

```
...
! Move up through the atmosphere computing the derivatives at each half-level
do jlev = nlev,1,-1
  ! Inline everything in one loop over g-points
  lw_deriv_old  = lw_deriv_g_region
  sum_tmp       = 0.0_jprb
  associate(A=>u_matrix(:,:,jlev+1), b=>lw_deriv_old)
    !$omp simd reduction(+:sum_tmp)
    do jg = 1, ng
      ! Compute effect of overlap at half-level jlev+1, yielding derivatives just above
      ! that half-level (matrix-vector multiply)
      ! both inner and outer loop of the matrix loops j1 and j2 unrolled
      ! inner loop:                 j2=1              j2=2              j2=3
      lw_deriv_g_region(jg,1) = A(1,1)*b(jg,1) + A(1,2)*b(jg,2) + A(1,3)*b(jg,3)
      lw_deriv_g_region(jg,2) = A(2,1)*b(jg,1) + A(2,2)*b(jg,2) + A(2,3)*b(jg,3)
      lw_deriv_g_region(jg,3) = A(3,1)*b(jg,1) + A(3,2)*b(jg,2) + A(3,3)*b(jg,3)

      ! Compute effect of transmittance of layer jlev, yielding
      ! derivatives just below the half-level above (jlev)
      lw_deriv_g_region(jg,1) = lw_deriv_g_region(jg,1) * transmittance(jg,1,jlev)
      lw_deriv_g_region(jg,2) = lw_deriv_g_region(jg,2) * transmittance(jg,2,jlev)
      lw_deriv_g_region(jg,3) = lw_deriv_g_region(jg,3) * transmittance(jg,3,jlev)

      sum_tmp = sum_tmp + lw_deriv_g_region(jg,1) + lw_deriv_g_region(jg,2) + &
            & + lw_deriv_g_region(jg,3)
    end do
  end associate

  lw_deriv(icol, jlev) = sum_tmp
end do
```

Figure A2. Reference (top) and optimized (bottom) version of the longwave derivatives kernel used by TripleClouds.

Data Availability Statement

The development version of ecRad 1.6, which includes our configurable optimizations and new gas optics schemes (ecCKD, RRTMGP and RRTMGP-NN), is available on Github and has also been archived on Zenodo (Ukkonen, 2022b). We expect most of the optimizations to feature in a future official version of ecRad, which is available on Github.

Acknowledgments
P.U. is grateful for the support from the European Commission through the projects "Energy-efficient SCalable Algorithms for weather and climate Prediction at Exascale" (ESCAPE-2, Grant ID 800897) and Destination Earth.

References

Chevallier, F., Chéruy, F., Scott, N., & Chédin, A. (1998). A neural network approach for a fast and accurate computation of a longwave radiative budget. *Journal of Applied Meteorology*, *37*(11), 1385–1397. https://doi.org/10.1175/1520-0450(1998)037<1385:annafa>2.0.co;2

Cotronei, A., & Slawig, T. (2020). Single-precision arithmetic in ECHAM radiation reduces runtime and energy consumption. *Geoscientific Model Development*, *13*(6), 2783–2804. https://doi.org/10.5194/gmd-13-2783-2020

Fielding, M. D., Schäfer, S. A., Hogan, R. J., & Forbes, R. M. (2020). Parametrizing cloud geometry and its application in a subgrid cloud-edge erosion scheme. *Quarterly Journal of the Royal Meteorological Society*, *146*(729), 1651–1667. https://doi.org/10.1002/qj.3758

Fu, Q., Liou, K., Cribb, M., Charlock, T., & Grossman, A. (1997). Multiple scattering parameterization in thermal infrared radiative transfer. *Journal of the Atmospheric Sciences*, *54*(24), 2799–2812. https://doi.org/10.1175/1520-0469(1997)054<2799:mspiti>2.0.co;2

Fuhrer, O., Chadha, T., Hoefler, T., Kwasniewski, G., Lapillonne, X., Leutwyler, D., et al. (2018). Near-global climate simulation at 1 km resolution: Establishing a performance baseline on 4888 GPUs with COSMO 5.0. *Geoscientific Model Development*, *11*(4), 1665–1681. https://doi.org/10.5194/gmd-11-1665-2018

Goody, R., West, R., Chen, L., & Crisp, D. (1989). The correlated-k method for radiation calculations in nonhomogeneous atmospheres. *Journal of Quantitative Spectroscopy and Radiative Transfer*, *42*(6), 539–550. https://doi.org/10.1016/0022-4073(89)90044-7

Hager, G., & Wellein, G. (2010). *Introduction to high performance computing for scientists and engineers*. CRC Press.

Higham, N. J. (2005). The scaling and squaring method for the matrix exponential revisited. *SIAM Journal on Matrix Analysis and Applications*, *26*(4), 1179–1193. https://doi.org/10.1137/04061101x

Hogan, R. J. (2010). The full-spectrum correlated-k method for longwave atmospheric radiative transfer using an effective Planck function. *Journal of the Atmospheric Sciences*, *67*(6), 2086–2100. https://doi.org/10.1175/2010jas3202.1

Hogan, R. J., & Bozzo, A. (2015). Mitigating errors in surface temperature forecasts using approximate radiation updates. *Journal of Advances in Modeling Earth Systems*, *7*(2), 836–853. https://doi.org/10.1002/2015ms000455

Hogan, R. J., & Bozzo, A. (2018). A flexible and efficient radiation scheme for the ECMWF model. *Journal of Advances in Modeling Earth Systems*, *10*(8), 1990–2008. https://doi.org/10.1029/2018MS001364

Hogan, R. J., Fielding, M. D., Barker, H. W., Villefranque, N., & Schäfer, S. A. (2019). Entrapment: An important mechanism to explain the shortwave 3D radiative effect of clouds. *Journal of the Atmospheric Sciences*, *76*(7), 2123–2141.

Hogan, R. J., & Matricardi, M. (2022). A tool for generating fast k-distribution gas-optics models for weather and climate applications. *Journal of Advances in Modeling Earth Systems*, *14*(10), e2022MS003033. https://doi.org/10.1029/2022MS003033

Hogan, R. J., Quaife, T., & Braghiere, R. (2018). Fast matrix treatment of 3-D radiative transfer in vegetation canopies: Spartacus-vegetation 1.1. *Geoscientific Model Development*, *11*(1), 339–350. https://doi.org/10.5194/gmd-11-339-2018

Hogan, R. J., Schäfer, S. A., Klinger, C., Chiu, J. C., & Mayer, B. (2016). Representing 3-D cloud radiation effects in two-stream schemes: 2. Matrix formulation and broadband evaluation. *Journal of Geophysical Research: Atmospheres*, *121*(14), 8583–8599. https://doi.org/10.1002/2016jd024875

Krasnopolsky, V. M., Fox-Rabinovitz, M. S., & Belochitski, A. A. (2008). Decadal climate simulations using accurate and fast neural network emulation of full, longwave and shortwave, radiation. *Monthly Weather Review*, *136*(10), 3683–3695. https://doi.org/10.1175/2008mwr2385.1

Lagerquist, R., Turner, D., Ebert-Uphoff, I., Stewart, J., & Hagerty, V. (2021). Using deep learning to emulate and accelerate a radiative transfer model. *Journal of Atmospheric and Oceanic Technology*, *38*(10), 1673–1696. https://doi.org/10.1175/jtech-d-21-0007.1

Meador, W., & Weaver, W. (1980). Two-stream approximations to radiative transfer in planetary atmospheres: A unified description of existing methods and a new improvement. *Journal of the Atmospheric Sciences*, *37*(3), 630–643. https://doi.org/10.1175/1520-0469(1980)037<0630:tsatrt>2.0.co;2

Michalakes, J., Iacono, M. J., & Jessup, E. R. (2016). Optimizing weather model radiative transfer physics for Intel's Many Integrated Core (MIC) architecture. *Parallel Processing Letters*, *26*(04), 1650019. https://doi.org/10.1142/s0129626416500195

Mlawer, E. J., Taubman, S. J., Brown, P. D., Iacono, M. J., & Clough, S. A. (1997). Radiative transfer for inhomogeneous atmospheres: RRTM, a validated correlated-k model for the longwave. *Journal of Geophysical Research*, *102*(D14), 16663–16682. https://doi.org/10.1029/97jd00237

Modest, M. F., & Zhang, H. (2002). The full-spectrum correlated-k distribution for thermal radiation from molecular gas-particulate mixtures. *Journal of Heat Transfer*, *124*(1), 30–38. https://doi.org/10.1115/1.1418697

Pal, A., Mahajan, S., & Norman, M. R. (2019). Using deep neural networks as cost-effective surrogate models for super-parameterized E3SM radiative transfer. *Geophysical Research Letters*, *46*(11), 6069–6079. https://doi.org/10.1029/2018GL081646

Pincus, R., Barker, H. W., & Morcrette, J.-J. (2003). A fast, flexible, approximate technique for computing radiative transfer in inhomogeneous cloud fields. *Journal of Geophysical Research*, *108*(D13), 4376. https://doi.org/10.1029/2002jd003322

Pincus, R., Mlawer, E. J., & Delamere, J. S. (2019). Balancing accuracy, efficiency, and flexibility in radiation calculations for dynamical models. *Journal of Advances in Modeling Earth Systems*, *11*(10), 3074–3089. https://doi.org/10.1029/2019ms001621

Schäfer, S. A. (2017). *What is the global impact of 3D cloud-radiation interactions?* (Doctoral dissertation). University of Reading. Retrieved from https://centaur.reading.ac.uk/72752/

Schäfer, S. A., Hogan, R. J., Klinger, C., Chiu, J. C., & Mayer, B. (2016). Representing 3-D cloud radiation effects in two-stream schemes: 1. Longwave considerations and effective cloud edge length. *Journal of Geophysical Research: Atmospheres*, *121*(14), 8567–8582. https://doi.org/10.1002/2016jd024876

Shonk, J. K., & Hogan, R. J. (2008). Tripleclouds: An efficient method for representing horizontal cloud inhomogeneity in 1D radiation schemes by using three regions at each height. *Journal of Climate*, *21*(11), 2352–2370. https://doi.org/10.1175/2007jcli1940.1

Song, H.-J., & Roh, S. (2021). Improved weather forecasting using neural network emulation for radiation parameterization. *Journal of Advances in Modeling Earth Systems*, *13*(10), e2021MS002609. https://doi.org/10.1029/2021MS002609

Tian, B., Fetzer, E. J., Kahn, B. H., Teixeira, J., Manning, E., & Hearty, T. (2013). Evaluating CMIP5 models using airs tropospheric air temperature and specific humidity climatology. *Journal of Geophysical Research: Atmospheres*, *118*(1), 114–134. https://doi.org/10.1002/jgrd.50117

Ukkonen, P. (2022a). Exploring pathways to more accurate machine learning emulation of atmospheric radiative transfer. *Journal of Advances in Modeling Earth Systems*, *14*(4), e2021MS002875. https://doi.org/10.1029/2021MS002875

Ukkonen, P. (2022b). Optimized version of the ecRad radiation scheme with new RRTMGP-NN gas optics [Software]. Zenodo. https://doi.org/10.5281/zenodo.7852526

Ukkonen, P., & Hogan, R. J. (2023). Implementation of a machine-learned gas optics parameterization in the ECMWF Integrated Forecasting System: RRTMGP-NN 2.0. *Geoscientific Model Development*, *16*(11), 3241–3261. https://doi.org/10.5194/gmd-16-3241-2023

Ukkonen, P., Pincus, R., Hogan, R. J., Nielsen, K. P., & Kaas, E. (2020). Accelerating radiation computations for dynamical models with targeted machine learning and code optimization. *Journal of Advances in Modeling Earth Systems*, *12*(12), e2020MS002226. https://doi.org/10.1029/2020MS002226

Veerman, M. A., Pincus, R., Stoffer, R., Van Leeuwen, C. M., Podareanu, D., & Van Heerwaarden, C. C. (2021). Predicting atmospheric optical properties for radiative transfer computations using neural networks. *Philosophical Transactions of the Royal Society A*, *379*(2194), 20200095. https://doi.org/10.1098/rsta.2020.0095

Yao, Y., Zhong, X., Zheng, Y., & Wang, Z. (2023). A physics-incorporated deep learning framework for parameterization of atmospheric radiative transfer. *Journal of Advances in Modeling Earth Systems*, *15*(5), e2022MS003445. https://doi.org/10.1029/2022ms003445

Zdunkowski, W. G., Welch, R. M., & Korb, G. (1980). Investigation of the structure of typical two-stream methods for the calculation of solar fluxes and heating rates in clouds. *Contributions to Atmospheric Physics*, *53*, 147–166.