University of Reading

Progress towards accelerating the unified model on hybrid multi-core systems

Conference or Workshop Item

Published Version

Zhang, W., Xu, M., Evans, K., Norman, M., Morales-Hernandez, M., Mahajan, S., Hill, A., Manners, J., Shipway, B. and Maynard, C. (2021) Progress towards accelerating the unified model on hybrid multi-core systems. In: PASC '21: Platform for Advanced Scientific Computing ConferencE, 5 - 9 July 2021, Geneva, Switzerland. doi: https://doi.org/10.1145/3468267.3470612 Available at https://centaur.reading.ac.uk/100834/

It is advisable to refer to the publisher's version if you intend to cite from the work. See Guidance on citing.

To link to this article DOI: http://dx.doi.org/10.1145/3468267.3470612

Publisher: Association for Computing Machinery

All outputs in CentAUR are protected by Intellectual Property Rights law, including copyright law. Copyright and IPR is retained by the creators or other copyright holders. Terms and conditions for use of this material are defined in the End User Agreement.

www.reading.ac.uk/centaur

University of Reading

CentAUR

Central Archive at the University of Reading

Reading's research outputs online

Progress Towards Accelerating the Unified Model on Hybrid Multi-Core Systems

Wei Zhang
w0z@ornl.gov
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA

Min Xu
xum1@ornl.gov
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA

Katherine Evans
evanskj@ornl.gov
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA

Matthew Norman
normanmr@ornl.gov
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA

Mario Morales-Hernandez
moraleshernm@ornl.gov
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA

Salil Mahajan
mahajans@ornl.gov
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA

Adrian Hill
adrian.hill@metoffice.gov.uv
Met Office
Exeter EX1 3PB, UK

James Manners
james.manners@metoffice.gov.uk
Met Office
Exeter EX1 3PB, UK

Ben Shipway
ben.shipway@metoffice.gov.uk
Met Office
Exeter EX1 3PB, UK

Maynard Christopher
christopher.maynard@metoffice.gov.uk
Met Office
Exeter EX1 3PB, UK
University of Reading
Reading, UK

ABSTRACT

The cloud microphysics scheme, CASIM, and the radiation scheme, SOCRATES, are two computationally intensive parts within the Met Office's Unified Model (UM). This study enables CASIM and SOCRATES to use accelerated multi-core systems for optimal computational performance of the UM. Using profiling to guide our efforts, we refactored the code for optimal threading and kernel arrangement and implemented OpenACC directives manually or through the CLAW source-to-source translator. Initial porting results achieved 10.02x and 9.25x speedup in CASIM and SOCRATES respectively on 1 GPU compared with 1 CPU core. A granular performance analysis of the strategy and bottlenecks are discussed. These improvements will enable UM to run on heterogeneous computers and a path forward for further improvements is provided.

CCS CONCEPTS

• **Applied computing** → **Earth and atmospheric sciences**; • **Computing methodologies** → *Massively parallel algorithms.*

KEYWORDS

Unified Model, CASIM, SOCRATES, GPU porting, OpenACC

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

PASC '21, July 5–9, 2021, Geneva, Switzerland

© 2021 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-8563-3/21/07.
https://doi.org/10.1145/3468267.3470612

ACM Reference Format:
Wei Zhang, Min Xu, Katherine Evans, Matthew Norman, Mario Morales-Hernandez, Salil Mahajan, Adrian Hill, James Manners, Ben Shipway, and Maynard Christopher. 2021. Progress Towards Accelerating the Unified Model on Hybrid Multi-Core Systems. In *Platform for Advanced Scientific Computing Conference (PASC '21), July 5–9, 2021, Geneva, Switzerland*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3468267.3470612

1 INTRODUCTION

The Unified Model (UM), developed by the United Kingdom Met Office (UKMO), is a widely used Earth system model for numerical weather and climate prediction. Over the last two decades, implementations of Message Passing Interface (MPI) and Open Multi-Processing (OpenMP) have been used to improve parallel model performance on Central Processing Units (CPU) [17]. However, UM computations are approaching a billion degrees-of-freedom with complex, multiphase physics calculations and are therefore incredibly time-consuming [12]. Further optimizing the model with finer resolution and more accurate simulations is always attractive for UM developers and users.

The massive parallelism provided by Graphic Processing Units (GPUs) is prompting developers to move their models to hybrid CPU-GPU systems. A number of climate and weather modeling centers have accelerated the computationally intensive portions (hot spots) of their models by porting them to GPUs to exploit data parallelism, generally with CUDA programming or directive-based approaches. As one example, GPU acceleration of the Weather Research Forecast (WRF) Single Moment 6-class (WSM6) microphysics

scheme obtained over 140x speedup compared with the CPU serial version [19]. Further optimization of WSM6 within the Model for Prediction Across Scales (MPAS) achieved 2.38x speedup for WSM6 on one GPU compared to 48 CPU cores [10]. Wang et al. [18] accelerated the Rapid Radiative Transfer Model for General circulation models (RRTMG) by about 18.52x on GPU compared with the CPU-based single-threaded version. Alvanos and Christoudias [1] implemented GPU capabilities into the chemical kinetics module in the global Atmosphere-Chemistry model ECHAM/MESSy (EMAC), and reached a relative speedup of 22x in their experiments.

Although performance hot spots ran much faster on GPU, the performance improvement was diluted when incorporated within an entire component and Earth system model. For example, the GPU acceleration of the Nucleus for European Modeling of the Ocean (NEMO) achieved a computation speedup up to 77x, but the total speedup was 50x [14]. Brown et al. [2] offloaded the entire Cloud AeroSol Interacting Microphysics (CASIM) package to the GPU using OpenACC. Their warm test case with 14,000 vertical columns ran 6.7x faster on GPU compared with running on one single CPU core. However, the overall speedup for the parent model, Met Office NERC Cloud Model (MONC), was only 1.4x. The performance improvement dilution was mainly due to the substantial portions of the model that were still running on CPU and the unavoidable data transfer between the host and accelerator. In order to alleviate the overhead of data exchange, some studies ported the models completely to GPU. One example is the GPU-accelerated Princeton Ocean Model (POM), of which the performance on 4 GPUs was equivalent to that on 408 CPU cores [20]. Another example is the complete porting of LASG/IAP climate system ocean model by Jiang et al. [9], which achieved a 6.6x speedup on 4 NVIDIA K80 cards compared with 4 Intel(R) Xeon(R) E5-2690 v2 CPUs. Therefore, in order to accelerate the whole model, it is important to find the optimal long-term strategy that not only reduces the computational time of the hot spots, but also minimizes the data transfers between the CPU and GPU.

Considering these results, we chose two important and computationally intensive physics components within the UM [17] to target for performance improvements. The Suite Of Community RAdiative Transfer codes based on Edwards and Slingo (SOCRATES) [7, 11] is the default radiative transfer model in UM and CASIM is a recent addition to UM cloud microphysics. CASIM performs a detailed simulation of aerosol effects and in-cloud aerosol processing. Based on the lessons learned from Brown et al. [2], we have enabled CASIM and SOCRATES to use the GPU still using OpenACC but with a more substantial refactoring strategy so as to more effectively use the GPU. We present the benefits and limitations of GPU acceleration of a refactored CASIM and SOCRATES as part of a long-term strategy with the goal of enabling the UM to utilize the accelerated node systems (CPU-GPU).

The rest of the paper is organized as follows. Section 2 describes the methodology and strategy of our GPU implementation. Section 3 and Section 4 present the detailed code refactorings, porting approaches and performance comparisons for CASIM and SOCRATES, respectively. Section 5 provides a conclusion of our current optimization work and discusses the future plan.

2 METHODOLOGY

The refactoring strategy to prepare CASIM and SOCRATES for parallel execution on GPU was based on two profiling tools to identify computational hot spots and compare performance of evolved code versions. The General Purpose Timing Library (GPTL) [15] displays detailed timing information, calling tree, and how many times each subroutine is called. The NVIDIA profiler, NVPROF, allows users to visualize the timeline of the application's activity on CPU and GPU. It can also detect potential performance bottlenecks and guide us to performance improvement opportunities.

Our strategy was to use OpenACC directives to offload the calculations to GPU and manage data transfer between CPU and GPU. The OpenACC is a directive-based approach for GPU porting and promises less complexity and programming effort compared with rewriting the entire program using low-level CUDA functions. Several climate models have used OpenACC and achieved acceptable speedup (e.g., the CAM-SE climate model in Norman et al. [13], the NICAM atmospheric model in Demeshko et al [6], and the LASG/IAP Climate System Ocean Model in Jiang et al. [9]). Along with manually implementing the OpenACC directives, we also utilized a single column abstraction (SCA) incorporation of the CLAW compiler where appropriate [4, 5] to perform OpenACC-specific code transformation for SOCRATES after we converted it to a single-column structure. The CLAW compiler is a source-to-source translator for FORTRAN codes. It transforms the CLAW directives, a domain-specific language (DSL), to either OpenMP or OpenACC directives to achieve performance portability in weather and climate models.

CASIM and SOCRATES are both serial codes that rely on their parent models, the UM and MONC, to provide parallelization. However, running UM or MONC is not efficient for agile development. Therefore, we used two simple serial models to drive CASIM and SOCRATES respectively and performed verification and performance against the CPU-based original codes. All test experiments were conducted using one NVIDIA V100 GPU and one IBM POWER9 CPU core on Summit supercomputer, which is housed by the Oak Ridge Leadership Facility (OLCF) at the Oak Ridge National Laboratory (ORNL). When compiling the model with PGI compiler, the flag '-fast' was used for automatic optimization, and '-Mipa=inline:reshape' was used for module inlining.

In all, our implementations of new code for GPU follows this development cycle within SOCRATES and CASIM:

(1) `Analyze`: Profile the codes to identify computationally intensive portions or performance bottlenecks.
(2) `Parallelize and optimize`: Refactor the code, reduce the bottlenecks, and implement OpenACC directives to offload calculations to GPU.
(3) `Validation and verification`: Insure the accuracy and robustness of modeling output and performance results.
(4) Repeat as new performance bottlenecks emerge to achieve further performance improvement.

3 THE GPU IMPLEMENTATION OF CASIM

The CASIM source code used in our experiments was cloned from branch vn0.3.3_kid_casim_optimise with the SVN revision number r6985 and has been tested within MONC and the Kinematic Driver

(KiD) [16]. This version has the vertical loops pushed down to the process routines, which is different from the one used by Brown et al. [2] in which the vertical loops were still in *microphysics_common* subroutine and all process rates were derived point wise.

The source code is written in FORTRAN90 with about 125 subroutines among 50 modules, and 16,300 lines in total. KiD was used as the parent model in our experiments. KiD is a serial kinematic framework constraining the dynamics and isolating the microphysics. It is publicly available at https://github.com/Adehill/KiD-A. Many test cases are provided in KiD default repository. We chose the most computationally intensive two-dimensional squall line case to verify our code refactoring and compare the performance. This case simulates lots of convective clouds. The benchmark simulation on the CPU was configured with 3200 columns and 52 vertical levels for 600 steps using 2-seconds time-step, which is a typical time-step for CASIM in LES or Kinematic. In the UM we would use different numbers of columns and vertical levels depending on the simulated regions, and longer time-steps. However, from experience refactoring code for CPU in MONC/KiD and applying it to the UM, the speedups tend to be transferable.

3.1 Initial code refactoring

CASIM simulates the microphysics for 6 different moisture states: vapor, cloud, rain, snow, ice and graupel. It also explicitly represents the effect and in-cloud processing of aerosol. Generally speaking, the microphysical processes can be divided to cold and warm microphysics. The cold microphysics are any processes that include snow, ice and graupel, including mixed phase clouds. These processes can occur below the freezing level, e.g. graupel falls to the surface and snow melts, while warm microphysics focuses on the mechanisms by which the cloud droplets are formed from vapor and further grow to raindrops. The parameterizations of aerosol physics and cold microphysics are still being actively developed and tested in the Met Office. Therefore, presently we focus on optimizing the simulations of warm microphysics. In our experiments, the cold microphysics is switched off and 5 prognostics are used (vapor, cloud mass and number, rain mass and number). However, the cold microphysics share the same loop structure with the warm microphysics. Therefore, we expect that the optimization resulting from parallelizing the loops can be extended to the whole moisture-states and we will include the optimization of cold micropysics in our future work.

The original calculations in CASIM are performed column by column. The allocated variables are declared for the single column on which CASIM is currently working. In order to work in parallel on GPU, each allocatable array was extended by two dimensions indexed by i and j coordinates representing the horizontal coordinates of that column. CASIM uses a derived data type (DDT) to hold the diagnostic tendencies of moisture in different states due to different microphysical processes. This DDT was flattened to a normal array to avoid the errors we met when transferring DDT between CPU and GPU. Additionally, *"!$acc declare create(arrays)"* was added under the declaration of all global variables to map them to the GPU and keep their residency during the full execution. The overview of this refactoring process is shown in Figure 1.

```
!! Original code-------------

real (wp), allocatable :: pressure(:)
real(wp), allocatable :: qfields(:, :)
type(process_rate), allocatable :: procs(:, :)
...

subroutine initialise_micromain()
   ...
   allocate(pressure(nz))
   allocate(qfields(nz, nq))

   allocate(procs(ntotalq, nprocs))

   do iproc = 1, nprocs
      do iq = 1, ntotalq
         allocate(procs(iq, iproc)%column_data(nz))
      end do
   end do
   ...
end subroutine initialise_micromain
```

```
!! Refactoring code-------------

real (wp), allocatable :: pressure(:, :, :)
real(wp), allocatable :: qfields(:, :, :, :)
real(wp), allocatable :: procs_flat(:, :, :, :, :)
...
!$acc declare create(pressure, qfields, procs_flat, ...)

subroutine initialise_micromain()
   ...
   allocate(pressure(nz, is:ie, js:je))
   allocate(qfields(nz, nq, is:ie, js:je))

   allocate(procs_flat(nz, ntotalq, nprocs, is:ie, js:je))
   ...
end subroutine initialise_micromain
```

Figure 1: Overview of initial code refactoring over arrays

3.2 Implementation of OpenACC

The GPTL profile of the benchmark simulation is shown in Figure 2 with the nested subroutine calling structure and the runtime of relatively expensive subroutines illustrated. The interface subroutine, *casim_interface*, allows the parent model to pass input variables to the entry point subroutine, *shipway_microphysic*, in which the i-, j-loops call the *microphysics_common* subroutine, which further calls all other CASIM subroutines using a n-loop. The i-, j-loops are the horizontal loops working over the columns, and are safe to be parallelized because each iteration is completely independent. The n-loop updates the diagnostic variables at each iteration until the moisture field reaches a converged steady state, hence it must run sequentially. The vertical k-loops are within the subroutines located at the lower level of the call tree. Most of them can be parallelized except the one in the *sedr_1M_2M* subroutine, which determines the sedimentation rate and has vertical data dependency.

The computationally intensive loops are deep in the call tree and called conditionally by the n-loop for each column. Significant code refactoring would be required to extract each one out, port them to GPU and keep the massively parallel GPU busy. For that purpose, we would have to reorder the loops by pulling the n-loop up and pushing the horizontal loops down. This would lead to substantial difficulties analyzing the data dependencies, keeping the conditionals, and maintaining the correctness of computation. Therefore, we offloaded the entire *microphysics_common* to GPU. However unlike Brown et al. [2], we have applied a two-level parallelism approach to further exploit the data parallelism within *microphysic_common* and all other subroutines that it calls.

The first step in implementing OpenACC directives to CASIM was to add parallel regions and simply use unified memory. We found that the overhead of using unified memory swamped the speedup achieved from parallelizing the loops in the cases with large domain sizes. Therefore, we moved to the next step by using OpenACC data directives to manage the data transfer between CPU and GPU. The OpenACC implementation to CASIM is shown in Figure 3 with pseudo code. The original nested horizontal looping is split into three parts. The first part inherits the variables from the parent model, and transfers the data from CPU to GPU through *"!$acc update device(list of variables)"*. The second part launches the kernel and offloads the computation in *microphysics_common*

subroutine to GPU. Since the related variables have been mapped to GPU with the *"!$acc declare create"* clause, the *"!$acc present"* clause is used to indicate the existence of the storage address on the GPU, which avoids duplicate data copies. The third part has *"!$acc update self(list of variables)"* to transfer data from GPU to CPU and transfer back the tendencies.

The horizontal loops calling *microphysics_common* subroutine are decorated with *"!$acc parallel loop"*, which generates a number of blocks in CUDA. The *"collapse(2)"* clause is used to merge two horizontal loops into one because they are completely independent. The n-loop is marked with *"!$acc loop seq"* so that it runs serially. The parallelizable inner loops and subroutines (e.g., *sum_procs*, *condevp*) are mapped to threads by *"!$acc loop vector collapse(number)"* and *"!$acc routine vector"*, respectively. The *sedr_1M_2M* subroutine is treated carefully with *"!$acc routine seq"* to denote the inner calculations are done sequentially over the vertical levels.

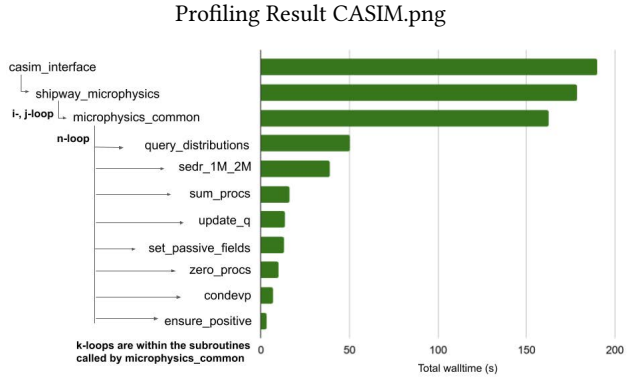


Figure 2: Overview of GPTL profiling results of KiD-CASIM benchmark case on CPU

```
subroutine shipway_microphysic()
  ...
  do j = js, je
     do i = is, ie
        qfields(:, i_qv, i, j) = qv_tmp(ks:ke, i, j)
        dqfields(:, i_qv, i, j) = dqv_tmp(ks:ke, i, j)
        ...
     end do
  end do
  !$acc update device(qfields, dqfields, …)

  !$acc parallel loop collapes(2) &
  !$acc present(qfields, dqfields, …)
  do j = js, je
     do i = is, ie
        call microphysics_common(i, j, qfields(:,:,i,j), &
dqfields(:,:,i,j), ..)
     end do
  end do

  !$acc update self(tend, …)
  do j = js, je
     do i = is, ie
        dqv_tmp(ks, ke, i, j) = tend(:, iqv, i, j)
     end do
   end do
   ...
end subroutine shipway_microphysics

subroutine microphysics_common(i, j, qfields, dqfields, …)
  …
  !$acc loop seq
  do n = 1, nsubsteps
     …
     call condevp(...)
     …
     call sedr_1M_2M(...)
     …
     call sum_procs(...)
     ....
  end do
  ....
end subroutine microphysics_common
```

```
subroutine condevp(...)
  !$acc routine vector
  ...
  !$acc loop vector
  do k = 1, nz
     ...
  end do
end subroutine condevp

subroutine sum_procs(...)
  !$acc routine vector
  ...
  !$acc loop vector collapse(3)
  do ip = 1, nprocs
     do iq = 1, nq
        do k = 1, nz
           ...
        end do
     end do
   end do
   ...
end subroutine sum_procs(...)

subroutine sedr_1M_2M(...)
  !$acc routine seq
  ...
  !$acc loop seq
  do k = nz-1, 1, 1
     ...
     dm = flux(k+1) - flux(k)
     ...
   end do
   ...
end subroutine sedr_1M_2M
```

Figure 3: Overview of OpenACC implementation on CASIM, showing pseudo code.

3.3 Results of the optimizations

A series of experiments over a range of domain sizes is conducted to evaluate the optimization at different problem size. Each experiment has the same number of vertical levels with the benchmark case, completes 600 time steps, and is repeated 10 times. The runtime varies 0.28-1.45% for GPU-CASIM with a range of 1.00-4.17%, and 0.16-0.51% for CPU-CASIM with a range of 0.45-1.76% on Summit. Similar fluctuations on Titan were observed by Evans et al. [8]. There is a small difference in comparing average versus best. For the most rigorous evaluation, we compared the performance using the averaged wall clock time of the 10 runs. The correctness of the modifications is verified by comparing simulation output using the same configuration on GPU versus CPU. The differences are visually indistinguishable in all results reported here and the root mean square errors are within the range of 1.0E-10 to 1.0E-7. These are considered acceptable to the physics model developers because they are within the range of the differences between the model results and observational data and are likely associated with roundoff differences when performing operations on the CPU or GPU.

Figure 4 indicates that all experiments achieved considerable speedup in the *microphysics_common* module within CASIM, where "speedup" is defined as the ratio of the averaged CPU runtime versus GPU runtime for each configuration. As expected, the speedup factor increases for larger domains, as GPU parallelization increases with problem size. However, the computation time for the entire CASIM called by the KiD interface, *mphys_casim*, does not decrease dramatically. NVPROF identifies that less than 5.17% of runtime on GPU is spent on data transfer between CPU and GPU. The API calls are dominated by `cuStreamSynchronize`, which is the time on the CPU side where the CPU is waiting for the GPU to finish the execution. Our porting did not involve asynchronization due to the data dependency between *microphysics_common* in CASIM and all other components left on the CPU, thus the execution on the CPU is blocked until the GPU has finished all issued tasks. However, this time is concurrent to the time that the code is running on the GPU. Therefore, it is not considered in performance comparison. The allocation and kernel launching expense is relatively small (e.g., about 4.25% for the experiment with 3200 columns), but is not negligible. To sum up, the overheads of data transfer, memory allocation and launching the kernels are the main reasons for less performance improvement achieved for the entire CASIM compared to that for the *microphysics_common* subroutine. Nevertheless, porting the entire *microphysics_common* shows substantial performance benefit by keeping the data transfer taking up only a small fraction of the overall time in all experiments.

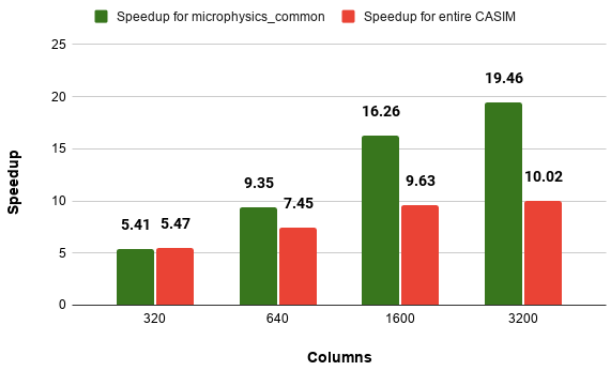


Figure 4: The GPU speedup for *microphysics_common* and the entire CASIM in experiments with different number of columns

4 THE GPU IMPLEMENTATION OF SOCRATES

SOCRATES contains both C and FORTRAN code, but the majority is FORTRAN-based and has been highly optimized for parallel, multi-core CPU computing. The source code was branched from SOCRATES for UM11.6 with the SVN revision number r855, which has 116 FORTRAN77 and 231 FORTRAN90 subroutines/functions, and about 124,018 lines total, including comments. The serial driver used for SOCRATES is *l_run_cdf*, which is an offline testing program that ingests meteorological and other data. SOCRATES has multiple scientific options for calculating short-wave (SW) and long-wave (LW) radiation. The random overlapping gas absorption scheme is the most accurate, but also the most computationally intensive scheme according to our performance profiling. Therefore, we target the "random overlap" scheme for GPU acceleration.

The cost of the scheme scales pretty linearly with the number of "monochromatic" calculations, i.e. the number of Exponential-Sum Fitting of Transmission (ESFT). The LW configuration generally has more monochromatic calculations than the SW configuration, which is why we saw a greater cost overall for the LW than the SW. Hence a test case for calculating LW radiation using the "random overlap" option is chosen for our experiments as the benchmark case. This case is provided in the test suite of SOCRATES's repository. However, we expect to achieve similar speedups in the SOCRATES SW case as in the LW case because LW and SW radiation schemes share most of the subroutines, and very few subroutines are solely used to calculate LW or SW radiation. Based on the profiling results of CPU-SOCRATES, these subroutines only take up a small amount of calculation time (i.e., *solar_coefficient_basic* and *mixed_solar_source* are the two subroutines that are only used in the SW calculation and account for 5.7% and 2.1% of the total time of *solve_band_random_overlap* respectively. Two other subroutines, *ir_source* and *adjust_ir_irradiance*, are only used in the LW calculation and use 2.2% and 4e-4% of the total time of *solve_band_random_overlap* respectively).

4.1 Initial code refactoring

SOCRATES employs many FORTRAN77 features such as *goto* and statement labels using numbers, which are obsolete and infeasible for GPU porting. So we rewrote relevant code following the FORTRAN90 standard and similar to the CASIM refactoring, flattened the derived data type variables needed for access on the GPU.

4.2 Implementation of OpenACC

For SOCRATES, we used GPTL to profile a benchmark simulation configured with 20 columns and 167 vertical levels. This configuration is derived from a field experiment simulating plenty of fairly realistic convective clouds in a Large Eddy Model. 167 levels is more than what is used in the operational UM but not beyond the bounds of what we might expect to use in the future. The current UM runs bundle groups of 16 columns together in "segments" to loop through with OpenMP as this seems to be an optimum size for efficiency, so 20 columns is not too far from this. Besides, this experiment is very efficient for us to verify results after code refactoring and GPU porting. It can also be used to measure the GPU performance in a small number of columns for some site simulations.

As shown in Figure 5, the computational expense of the entire SOCRATES is dominated by the *solve_band_random_overlap* subroutine for gas absorption, within which the *mix_column* subroutine is the main culprit because it contains the treatment of the vertical overlap between different cloudy layers and is within the ESFT loop. Three hot spots are identified under the *mix_column* subroutine. They are *trans_source_coeff*, *two_coeff_cloud*, and *solver_mix_direct*, which consume 27%, 12% and 36% of the total runtime of *radiance_calc*, respectively. The *two_coeff_basic* and *ir_source* subroutines are also included as targets for GPU optimization because they have many parallelizable loops.

Unlike the GPU-based CASIM, the GPU-based SOCRATES in present work utilizes the unified memory to share the data between the CPU and GPU. Therefore, no explicit treatment is needed for allocating and copying device memory, and this allows us to focus on optimizing loops within the hot spots. Two approaches are used to reorganize the loops and explore their GPU parallelism, thus generating two versions of GPU-based SOCRATES for evaluation.

The first approach we term the "PUSHUP-SOCRATES" method as we pushed horizontal loops under the *solve_band_random_overlap* up to the *radiance_calc* and the *solve_band_random_overlap* became a single column model. This strategy was enacted so that we could incorporate the CLAW SCA (see the left panel of Figure 6). This method involved heavy refactoring of the original SOCRATES code including array demotion, argument change, subroutine modularization (where a subroutine is encapsulated into a FORTRAN90 module), and loop removal. After that, as illustrated in the right panel of Figure 6, the OpenACC implementation was performed automatically by the CLAW compiler, which pushed the horizontal loop back to the *solve_band_random_overlap* subroutine and parallelized the inner horizontal and ESFT loops. The vertical loops were not parallelized because the CLAW compiler could not determine their data dependencies. All subroutines called by the *solve_band_random_overlap* subroutine were treated as sequential OpenACC routines with the *"!$acc routine seq"* directive added by the CLAW compiler. In this approach, one big kernel over the ESFT

and horizontal loops is generated. This kernel contains 5 levels of nested subroutines and a total of 21 subroutines.

The second approach is termed the "PUSHDOWN-SOCRATES" method, in which the ESFT loop in the *solve_band_random_overlap* routine was pushed down and combined with vertical and horizontal loops in subroutines at a lower level. In this case, the OpenACC directives were implemented manually to parallelize the modified loops within the hot spots. As one example, Figure 7 shows the pseudo code blocks in the *trans_source_coeff* subroutine after the ESFT loop was pushed down into it (left panel) and an OpenACC kernel (parallel region) was created (right panel). There is a three-level tightly nested loop in the block and two expensive intrinsic functions (square-root and exponential functions) in the loop. Therefore, the OpenACC parallel region was created around this loop. In total, five kernels in five subroutines are generated. They are *two_coeff_basic*, *trans_source_coeff*, *ir_source*, *two_coeff_cloud*, and *solver_mix_direct*, respectively.

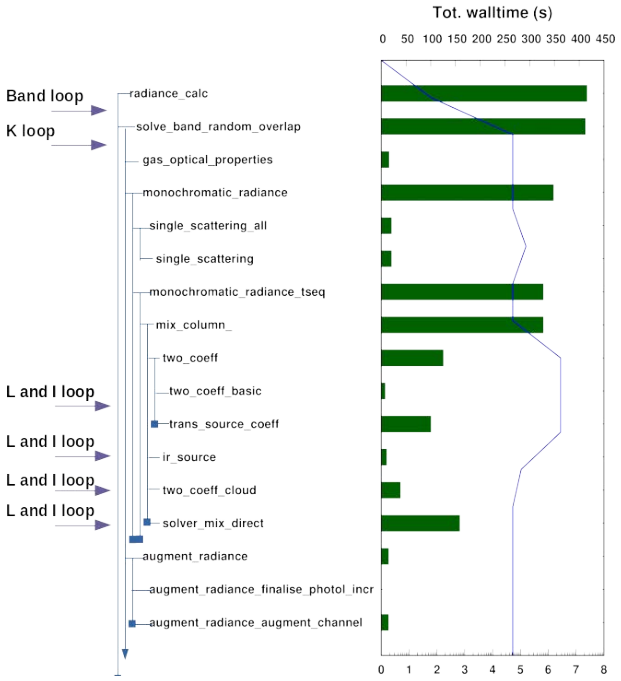


Figure 5: Overview of GPTL profiling results of *l_run_cdf* SOCRATES benchmark case on CPU with the blue line showing the number of subroutine calls using 1.E4 as units. (Band: spectral band loop, K: Exponential-Sum Fitting of Transmission (ESFT) loop, L: horizontal loop, and I: vertical loop)

4.3 Results of the optimizations

Two experiments with different spatial domain sizes are used to evaluate SOCRATES performance on a single CPU core and a GPU. The first experiment represents a simple field experiment with 20 columns and 167 vertical levels. The second is based on a UM global simulation in which the entire domain is spatially decomposed into 205 chunks. There are 1500 columns/grids in each chunk. The

```
!! codes after pushing up the horizontal loop (soc_hori)
!! in radiance_calc subroutine
SUBROUTINE radiance_calc(...)
  DO i_band=control%first_band, control%last_band
    ...
    !$claw sca forward
    DO soc_hori = 1, nd_profile
      CALL solve_band_random_overlap(..., soc_hori)
    END DO
  ...
  END DO
END SUBROUTINE radiance_calc

!! in solve_band_random_overlap
SUBROUTINE solve_band_random_overlap(..., nd_profile, nd_layer)
  ...
  !$claw define dimension icol (1:nd_profile)&
  !$claw sca
  DO k = 1, n_term
    ...
    CALL gas_optical_properties(...)
    CALL monochromatic_radiance(...)
    CALL augment_radiance(...)
    ...
  END DO
END SUBROUTINE solve_band_random_overlap
```

```
!! codes after complied the CLAW compiler
!! in radiance_calc subroutine
SUBROUTINE radiance_calc(...)
  DO i_band=control%first_band, control%last_band
    ...
    CALL solve_band_random_overlap(...)
    ...
  END DO
END SUBROUTINE radiance_calc

!! in solve_band_random_overlap
SUBROUTINE solve_band_random_overlap(..., nd_profile, nd_lay
  ...
  !$acc data present(cld_frac_cloud,cld_w_cloud,...)
  !$acc parallel
  !$acc loop gang vector &
  !$acc private(cg_coeff,uplm_zero,uplm_sol,k_esft ...)
  DO soc_hori = 1, nd_profile    !horizontal loop is pushed
    DO k = 1, n_term
      ...
      CALL gas_optical_properties(...)
      CALL monochromatic_radiance(...)
      CALL augment_radiance(...)
      ...
    END DO
  END DO
END SUBROUTINE solve_band_random_overlap
```

Figure 6: Overview of the implementation of the CLAW SCA on the single column *solve_band_random_overlap* subroutine of SOCRATES

```
!! codes after pushing down the k-loop
SUBROUTINE trans_source_coeff
  ...
  DO k =1, n_term
    DO i=i_layer_first, i_layer_last
      DO l=1, n_profile
        ...
        xlambda = SQRT(sum(l,i,k) * diff(l,i,k))
        exponential=exp(-xlambda*tau(l,i,k))
        exponential2=exponential*exponential
        ...
      END DO
    END DO
  END DO
END SUBROUTINE trans_source_coeff
```

```
!! codes after pushing down the k-loop
SUBROUTINE trans_source_coeff
  ...
  !$acc parallel loop gang vector collapse(3)
  DO k =1, n_term
    DO i=i_layer_first, i_layer_last
      DO l=1, n_profile
        ...
        xlambda = SQRT(sum(l,i,k) * diff(l,i,k))
        exponential=exp(-xlambda*tau(l,i,k))
        exponential2=exponential*exponential
        ...
      END DO
    END DO
  END DO
  !$acc end parallel
END SUBROUTINE trans_source_coeff
```

Figure 7: Overview of OpenACC implementation on loops in the *trans_source_coeff* subroutine of SOCRATES

computational cost of SOCRATES highly depends on the input data, especially cloud states. Following the experience of developing SOCRATES in Met Office, we chose 5 chunks at random by a visual inspection to efficiently pick up a mix of cloud to account for the effects of different cloud amounts and vertical distributions on the performance.

As with the CASIM performance results, the definition of the speedup factor is also the same. Again, the mean of results from 10 runs were used to remove the performance fluctuations even though they are relatively small (i.e., in the global experiment with 1500 columns, we found the variations are 0.25-1.63% for GPU-SOCRATES with a range of 0.82-4.81%, and 0.12-0.24% for CPU-SOCRATES with a range of 0.45-0.81%).

The speedup of the field experiment as a function of gangs, workers, and vectors for the PUSHUP method is shown in Figure 8 and indicates poor performance on the GPU. As there is only one large kernel, the number of gangs, workers, and vectors are the only tuning parameters. Given the 4 workers and 32 vectors, increasing the number of gangs from 1 to 80 leads to an increase of GPU speedup from 0.52 to 1.07. Further increasing the number of gangs from 80 to 160 does not get any performance improvement. There is no clear trend in the GPU speedup with the increase in the number of workers or vectors. The best performance is achieved by 80 gangs, 2 workers and 32 vectors, and is only 1.32x faster than the serial run on the CPU. Further increasing the number of gangs and workers leads to an out of memory (OOM) error because the big

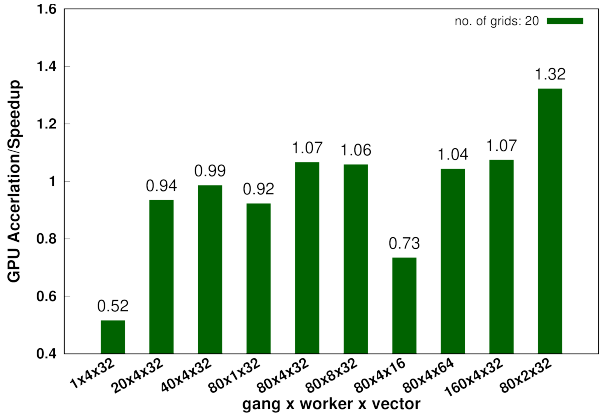
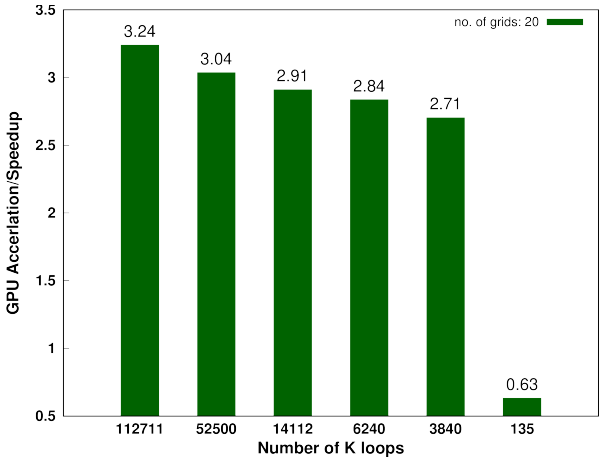


Figure 8: GPU speedups as a function of the number of gangs, workers and vectors (gang×worker×vector)

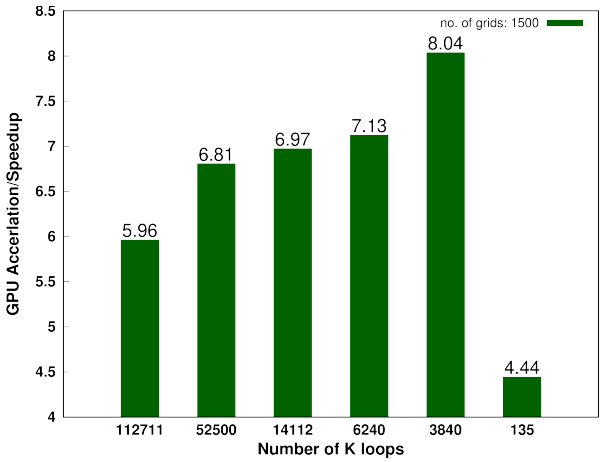
single kernel created in PUSHUP method quickly consumes the GPU memory. We found similar results in the global experiment.

Figure 9 illustrates how GPU acceleration varies with the number of ESFT loops (also called as K loop in SOCRATES) for the PUSHDOWN method for the field and global experiments. In the field experiment with 20 columns, the GPU accelerations increase from 2.71 to 3.24 as the number of K loops increases from 3840 to 112711. This is because more loops in the kernels will be computed in parallel by GPUs when the number of K loops increases and the data movement and memory usage in this experiment is not large enough to inhibit the performance enhancement. On the contrary, in the global experiment with 1500 columns, the data movement and memory usage are huge and easy to reach OOM errors. In order to reduce the risk of OOM, the K loop is unrolled into several chunks. Based on several tests, the block size is selected as 200 to achieve the best performance. The loop unrolling increases the number of subroutine calls, kernel launching, temporary array allocation/deallocation, and GPU faulting when the number of K loops increases. Therefore, as the number of K loops increases, the speedup achieved in the global experiment decreases. When K=135, the unrolling is not applied, and there is not much calculation to efficiently use the GPU, so only 4.44 speedup is achieved. Performance of individual kernels is also evaluated and shown in Figure 10. Most kernels have achieved remarkable speedup, especially the kernels in *trans_source_coeff* and *solver_mix_direct*, with 15.87x and 12.57x, respectively. Although the GPU kernel in *two_coeff_basic* is slightly slower, the average speedup for the *solve_band_random_overlap* is about 9.25x. The NVPROF profiling results show that data migrations of the unified memory for the PUSHDOWN-SOCRATES dominates 5.6-9.7% of total time of OpenACC kernels depending on the size of horizontal and ESFT loops. Less than 6% of total time is spent on launching kernels. The overheads for GPU page faults of the unified memory, however, can be as high as 49.2% of total time when the size of horizontal and ESFT loops is large. Explicitly implementing data directives or using CUDA data prefetching is a feasible method and will become part of our future work.

Overall, the acceleration obtained in PUSHUP-SOCRATES is much less than that in PUSHDOWN-SOCRATES. For the PUSHUP-SOCRATES, the big kernel generated by the CLAW compiler includes multiple levels of nested subroutines, which in our experiments easily reaches the bandwidth of memory, and causes an OOM error. Besides, some non-optimal tasks other than parallelizing loops are done in this big kernel so only a small speedup is promised by the PUSHUP method. In PUSHDOWN-SOCRATES, small kernels with more efficient parallelism are used, hence reaching better performance improvement.



(a) 20 columns



(b) 1500 columns

Figure 9: The GPU accelerations of the subroutine monochromatic_radiance vary with the number of ESFT (K) loops for two experiments of (a) 20 columns and (b) 1500 columns. All kernels for GPU optimization are under that subroutine.

5 CONCLUSION AND FUTURE WORK

The cloud microphysics package, CASIM, and the radiation package, SOCRATES, are two of the most computationally intensive

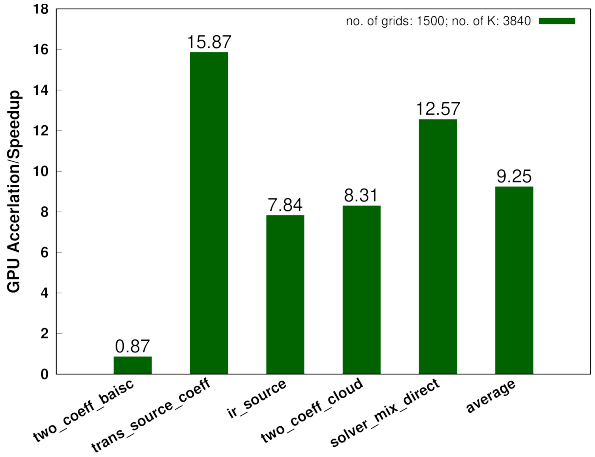


Figure 10: The GPU accelerations of five kernels and their average when the number of ESFT loops is 3840 for the experiment of 1500 columns.

portions of UM weather and climate simulations. We have presented a refactoring and OpenACC implementation of CASIM and SOCRATES for acceleration using NVIDIA GPU on the OLCF's Summit supercomputer. Considerable speedup has been achieved within targeted expensive kernels as well as the entire CASIM and PUSHDOWN-SOCRATES modules. It is important to note that the speedup is discussed by comparing the performance on one GPU against one CPU core. The reason is that we only use the serial model drivers for CASIM and SOCRATES on Summit at present. We will implement the accelerated CASIM and SOCRATES to UM, then enable the OpenACC build of CASIM and SOCRATES to be executed on multiple GPUs using MPI, and compare the best performance achieved by multiple GPUs and CPU cores.

The PUSHUP-SOCRATES strategy illustrated that heavy code refactoring to organize the SOCRATES code into a single-column structure was required so that CLAW could be applied. Although time-consuming, it was still worthwhile as a learning experience, as the CLAW-translated code provided good guidance to manual implementation. The performance enhancements achieved in PUSHUP-SOCRATES also motivate us to work closely with the CLAW developer and try other implementations of the CLAW compiler in our single column version of SOCRATES to further improve its performance in the future. The PUSHDOWN method also requires significant code refactoring, although it is a little less than the PUSHUP method. After we pushed down the ESFT loop and combined loops and created kernels in deep subroutines manually, we got a larger speedup compared to the PUSHUP method. Therefore it is not obvious that one strategy beats the other "hands down". For now, our strategy is to proceed by combining automatic and manual implementations, with efforts toward informing and improving the DSL strategy so that it may be the better-long terms solution once mature.

We also learned that calling multiple levels of nested subroutines in one big kernel could cause large overhead and easily use up the GPU registers and memory. Therefore, after pushing the loops down and using small kernels, more acceleration was achieved in SOCRATES. We will also apply this PUSHDOWN method to CASIM in the future.

Although the GPU-based CASIM and SOCRATES are running for several test cases on the GPU, there is still work to optimize the use of a large CPU-GPU computer like Summit. As kernels are optimized, more profiling will detect new performance bottlenecks and thus opportunities for further performance improvement. An important area of interest is to optimize memory access, for example, better use of fast registers and caches in the GPU memory hierarchy. In particular, both CASIM and SOCRATES have a large number of arrays being read several times from the global memory in multiple loops or subroutines, which takes time. Loop fusion and function fusion to merge several loops or subroutines will allow the data to be first read from the global memory and then read repeatedly from a register. Scalar replacement is also being considered to replace intermediate variable arrays as scalars that can be simply stored in the registers. Another important area is to mitigate the thread divergence due to *if* statements. In Chakroun et al. [3], the performance on GPU increased by 10-29% when applying refactoring compared with the original *if-then-else* instruction, which is very encouraging. Reducing the number of *if* statements is also necessary for us to get further optimization. However, careful and time-consuming analysis and significant code refactoring are needed and will be a focus in our future work. Specifically for SOCRATES, further performance improvement can be achieved by managing the data locality with explicit OpenACC data directives and increasing memory coalescing. It is also possible to increase the concurrency of launching kernels and the overlaps of kernel and memcpy, i.e. the kernels in clear-sky and cloud-sky radiation calculation can be launched asynchronously, and SW and LW radiation can be computed in different GPUs simultaneously. It is also necessary to adjust the number of gangs, workers, and the vector length, but the optimal values are highly dependent on the specific hardware and experiment.

For both CASIM and SOCRATES, we have not yet investigated every aspect, for example the individual CPU and GPU capabilities and memory bandwidth constrains. But they will be a focus during our next steps. Besides, our work will not be limited to use only OpenACC. We also plan to use OpenMP 4.5 to offload computations to GPU and compare the performance difference.

ACKNOWLEDGMENTS

This research was supported by the U.S. Air Force LCMC collaboration with Oak Ridge National Laboratory (ORNL). The computational resources on Summit are provided by the Oak Ridge Leadership Computing Facility (OLCF) Director's Discretion Project ATM112. The OLCF at Oak Ridge National Laboratory (ORNL) is supported by the Office of Science of the U.S. Department of Energy under Contract No.DE-AC05-00OR22725. Furthermore, we would like to acknowledge the contributions of Youngsung Kim at ORNL for the insightful suggestions on porting algorithm development and performance bottleneck detection. We also appreciate the great help on CLAW implementation from Valentim Clement at ORNL.

REFERENCES

[1] Michail Alvanos and Theodoros Christoudias. 2019. Accelerating Atmospheric Chemical Kinetics for Climate Simulations. *IEEE Transactions on Parallel and*

Distributed Systems 30, 11 (2019), 2396–2407.

[2] Nick Brown, Alexandr Nigay, Michèle Weiland, Adrian Hill, and Ben Shipway. 2020. Porting the microphysics model CASIM to GPU and KNL Cray machines. *arXiv preprint arXiv:2010.14823* (2020).

[3] Imen Chakroun, Mohand Mezmaz, Nouredine Melab, and Ahcene Bendjoudi. 2013. Reducing thread divergence in a GPU-accelerated branch-and-bound algorithm. *Concurrency and Computation: Practice and Experience* 25, 8 (2013), 1121–1136.

[4] Valentin Clement, Sylvaine Ferrachat, Oliver Fuhrer, Xavier Lapillonne, Carlos E. Osuna, Robert Pincus, Jon Rood, and William Sawyer. 2018. The CLAW DSL: Abstractions for Performance Portable Weather and Climate Models. In *Proceedings of the Platform for Advanced Scientific Computing Conference on - PASC '18*. ACM Press, Basel, Switzerland, 1–10. https://doi.org/10.1145/3218176.3218226

[5] Valentin Clement, Philippe Marti, Xavier Lapillonne, Oliver Fuhrer, and William Sawyer. 2019. Automatic Port to OpenACC/OpenMP for Physical Parameterization in Climate and Weather Code Using the CLAW Compiler. *Supercomputing Frontiers and Innovations* 6, 3 (2019), 51–63. https://superfri.org/superfri/article/view/285

[6] Irina Demeshko, Naoya Maruyama, Hirofumi Tomita, and Satoshi Matsuoka. 2012. Multi-GPU implementation of the NICAM atmospheric model. In *European Conference on Parallel Processing*. Springer, 175–184.

[7] J. M. Edwards and A. Slingo. 1996. Studies with a flexible new radiation code. I: Choosing a configuration for a large-scale model. *Quarterly Journal of the Royal Meteorological Society* 122, 531 (April 1996), 689–719. https://doi.org/10.1002/qj.49712253107

[8] Katherine J Evans, Richard K Archibald, David J Gardner, Matthew R Norman, Mark A Taylor, Carol S Woodward, and Patrick H Worley. 2019. Performance analysis of fully explicit and fully implicit solvers within a spectral element shallow-water atmosphere model. *The International Journal of High Performance Computing Applications* 33, 2 (March 2019), 268–284. https://doi.org/10.1177/1094342017736373

[9] Jinrong Jiang, Pengfei Lin, Joey Wang, Hailong Liu, Xuebin Chi, Huiqun Hao, Yuzhu Wang, Wu Wang, and Linghan Zhang. 2019. Porting LASG/IAP Climate System Ocean Model to Gpus Using OpenAcc. *IEEE Access* 7 (2019), 154490–154501.

[10] Jae Youp Kim, Ji-Sun Kang, and Minsu Joh. 2020. GPU acceleration of MPAS microphysics WSM6 using OpenACC directives: Performance and verification. *Computers & Geosciences* (2020), 104627.

[11] James Manners, John M. Edwards, Peter Hill, and Jean-Claude Thelen. 2015. SOCRATES (Suite Of Community RAdiative Transfer codes based on Edwards and Slingo) technical guide. https://code.metoffice.gov.uk/trac/socrates

[12] Christopher M Maynard and David N Walters. 2019. Mixed-precision arithmetic in the ENDGame dynamical core of the Unified Model, a numerical weather prediction and climate model code. *Computer Physics Communications* 244 (2019), 69–75.

[13] Matthew Norman, Jeffrey Larkin, Aaron Vose, and Katherine Evans. 2015. A case study of CUDA FORTRAN and OpenACC for an atmospheric climate kernel. *Journal of computational science* 9 (2015), 1–6.

[14] Sergi Palomas Martinez. 2019. Accelerating Operational Earth System Models using GPUs: portability of NEMO diagnostics to GPU's. (2019).

[15] Jim Rosinski. 2009. General purpose timing library (gptl): A tool for characterizing performance of parallel and serial applications. In *Cray User Group (CUG)*. Berkley, California.

[16] BJ Shipway and AA Hill. 2011. The Kinematic Driver model (KiD). *Technical Re* (2011).

[17] Karthee Sivalingam, Grenville Lister, and Bryan Lawrence. 2015. Performance analysis and Optimisation of the Met Unified Model on a Cray XC30. *arXiv preprint arXiv:1511.03885* (2015).

[18] Yuzhu Wang, Yuan Zhao, Wei Li, Jinrong Jiang, Xiaohui Ji, and Albert Y Zomaya. 2019. Using a GPU to accelerate a longwave radiative transfer model with efficient CUDA-based methods. *Applied Sciences* 9, 19 (2019), 4039.

[19] Huadong Xiao, Jing Sun, Xiaofeng Bian, and Zhijun Dai. 2013. GPU acceleration of the WSM6 cloud microphysics scheme in GRAPES model. *Computers & Geosciences* 59 (2013), 156–162.

[20] Shizhen Xu, Xiaomeng Huang, L-Y Oey, Fanghua Xu, Haohuan Fu, Yan Zhang, and Guangwen Yang. 2015. POM. gpu-v1. 0: a GPU-based Princeton Ocean Model. *Geoscientific Model Development* 8, 9 (2015), 2815–2827.