JAMES | Journal of Advances in Modeling Earth Systems®

RESEARCH ARTICLE
10.1029/2022MS003515

Acceleration of the Parameterization of Unified Microphysics Across Scales (PUMAS) on the Graphics Processing Unit (GPU) With Directive-Based Methods

Jian Sun[1], John M. Dennis[1], Sheri A. Mickelson[1], Brian Vanderwende[1], Andrew Gettelman[1,2], and Katherine Thayer-Calder[1]

[1]National Center for Atmospheric Research, Boulder, CO, USA, [2]Now at Pacific Northwest National Laboratory, Richland, WA, USA

Key Points:

- OpenACC and OpenMP target offload achieve competitive performance on the graphics processing unit (GPU), but OpenACC performs better for Parameterization of Unified Microphysics Across Scales
- GPU can outperform CPU in a practical Community Atmosphere Model simulation and using multiple GPUs per node is beneficial, especially at a high resolution
- The data transfer between CPU and GPU and the data allocation on GPU dominate the overall GPU performance

Supporting Information:

Supporting Information may be found in the online version of this article.

Correspondence to:

J. Sun and J. M. Dennis,
sunjian@ucar.edu;
dennis@ucar.edu

Citation:

Sun, J., Dennis, J. M., Mickelson, S. A., Vanderwende, B., Gettelman, A., & Thayer-Calder, K. (2023). Acceleration of the Parameterization of Unified Microphysics Across Scales (PUMAS) on the graphics processing unit (GPU) with directive-based methods. *Journal of Advances in Modeling Earth Systems*, *15*, e2022MS003515. https://doi.org/10.1029/2022MS003515

Received 23 NOV 2022
Accepted 10 MAY 2023
Corrected 16 JUN 2023

This article was corrected on 16 JUN 2023. See the end of the full text for details.

© 2023 The Authors. Journal of Advances in Modeling Earth Systems published by Wiley Periodicals LLC on behalf of American Geophysical Union. This is an open access article under the terms of the Creative Commons Attribution-NonCommercial-NoDerivs License, which permits use and distribution in any medium, provided the original work is properly cited, the use is non-commercial and no modifications or adaptations are made.

Abstract Cloud microphysics is one of the most time-consuming components in a climate model. In this study, we port the cloud microphysics parameterization in the Community Atmosphere Model (CAM), known as Parameterization of Unified Microphysics Across Scales (PUMAS), from CPU to GPU to seek a computational speedup. The directive-based methods (OpenACC and OpenMP target offload) are determined as the best fit specifically for our development practices, which enable a single version of source code to run either on the CPU or GPU, and yield a better portability and maintainability. Their performance is first examined in a PUMAS stand-alone kernel and the directive-based methods can outperform a CPU node as long as there is enough computational burden on the GPU. A consistent behavior is observed when we run PUMAS on the GPU in a practical CAM simulation. A 3.6× speedup of the PUMAS execution time, including data movement between CPU and GPU, is achieved at a coarse horizontal resolution (8 NVIDIA V100 GPUs against 36 Intel Skylake CPU cores). This speedup further increases up to 5.4× at a high resolution (24 NVIDIA V100 GPUs against 108 Intel Skylake CPU cores), which highlights the fact that GPU favors larger problem size. This study demonstrates that using GPU in a CAM simulation can save noticeable computational costs even with a small portion of code being GPU-enabled. Therefore, we are encouraged to port more parameterizations to GPU to take advantage of its computational benefit.

Plain Language Summary Cloud microphysics is an important process in the atmosphere that describes the formation, growth, evaporation, and melting of cloud and precipitation particles. Therefore, it is a critical component for a weather or climate model to produce a reasonable prediction of cloud and precipitation. Due to its complexity, it is typically time-consuming to compute the cloud microphysics. In this study, we enable the execution of cloud microphysics on the graphics processing unit (GPU), which is a processor that can run more computations simultaneously than a traditional CPU. There are multiple ways to program on GPU and we select the directive-based method. It allows the same code to run on either the CPU or GPU without changing the original programming language. We demonstrate that by porting the cloud microphysics to GPU, we are able to the achieve noticeable speedup against CPU. Furthermore, the speedup is higher if we could take advantage of multiple GPUs at the same time or lay a heavier computation on each GPU.

1. Introduction

Cloud microphysics is a microscale (from sub-micron to cm) process in the atmosphere, which affects the formation and evolution of cloud and precipitation particles. It interacts closely with cloud macrophysics, radiation and aerosol microphysics, and has a direct effect on the atmospheric water and energy budgets (Gettelman & Morrison, 2015; Gettelman et al., 2008, 2015, 2019; Morrison & Gettelman, 2008). However, a big challenge of representing cloud microphysics in a global climate model is its complexity and small scale (Morrison et al., 2009, 2020). Therefore, a parameterization is usually implemented in order to reflect the unresolved-scale features on the resolved-scale model variables and it is typically computationally expensive. For example, the cloud microphysics parameterization in the Community Atmosphere Model (CAM), known as Parameterization of Unified Microphysics Across Scales (PUMAS), contributes about 8% of total computational time of CAM physics but only represents about 1.2% of the total CAM physics code volume. Considering the fact that most global climate models have already been highly parallelized on a conventional multi-core architecture within a node through the shared-memory computing (e.g., Open MultiProcessor (OpenMP)) and across nodes through

the distributed-memory computing (e.g., Message Passing Interface (MPI)), we need to explore additional techniques in order to further speed up the computation of cloud microphysics.

Graphics Processing Unit (GPU) computing is a computational architecture that efficiently supports massive parallelism. In addition, heterogeneous architectures, meaning that there are both CPU and GPU available on the same compute node, are becoming more common on modern supercomputers, and GPUs are gradually contributing more computational performance than CPUs (e.g., the Summit supercomputer at Oak Ridge National Laboratory). Therefore, an obvious question would be raised: whether atmospheric modeling could benefit from GPU computing? The answer is yes as many atmospheric problems are highly independent and scalable in the spatial dimensions. Several related studies have shown a promising speedup by exploiting GPU. Mielikainen et al. (2013) used CUDA to port the Weather Research and Forecasting model (WRF) double-moment 6-class microphysics scheme (WDM6) to GPU and a single NVIDIA GeForce GTX 590 GPU was able to reach 150× speedup against one Intel Core i7 970 CPU core. Fuhrer et al. (2018) rewrote the dynamical core of the Consortium for Small-Scale Modeling (COSMO) model in C++ and used the C++ template library-based domain-specific language (DSL) for GPU offload. They found that using a single P100 GPU per node was able to achieve significantly higher simulated years per wall clock day (SYPD) compared to 12 Intel Haswell CPU cores per node. Sun et al. (2018) developed a box model of the gas-phase chemistry in CAM4-Chem, where the source code of the chemistry solver was separated from the host model, and thus could be compiled and computed quickly. They then ported the CPU-based box model to GPU using CUDA and OpenACC. They found out that using a single K20 GPU was able to achieve 1.33× speedup compared to 16 AMD Opteron™ 6274 CPU cores and the performance of OpenACC and CUDA was close when the problem size is sufficiently large. Wang et al. (2019) used CUDA Fortran to port the Rapid Radiative Transfer Model for General circulation models (RRTMG) longwave radiation scheme to GPU and a single K40 GPU was able to achieve 18.52× speedup compared to a single Intel Xeon E5-2680 CPU core. Bertagna et al. (2019) rewrote the High-Order Methods Modeling Environment (HOMME) dynamical core of the Energy Exascale Earth System Model (E3SM) in C++ and used the Kokkos library to achieve on-node parallelism and GPU offload. Their results showed that the performance of a single V100 GPU was about 1.2×–3.8× faster than a single CPU node (24–68 Intel CPU cores per node). J. Y. Kim et al. (2021) used OpenACC to port the Weather Research and Forecasting (WRF) single-moment 6-class microphysics scheme (WSM6) embedded within the Model for Prediction Across Scales (MPAS) to GPU. By using one V100 GPU, they could obtain a 2.38× speedup compared to 48 Intel Skylake CPU cores. Recently the Energy Exascale Earth System Model (E3SM) Atmosphere Model (EAM) was rewritten from Fortran to C++ (known as EAMxx). The Kokkos library was later used to achieve performance portability on CPUs and GPUs. Other similar efforts of GPU enablement of weather and climate models or components could be found in Norman et al. (2015), Fu et al. (2016), Mielikainen et al. (2016), and Abdi et al. (2019). The calculation of cloud microphysics parameterization is similar to that of atmospheric chemistry and radiation because it is mostly grid-independent in a global climate model. Hence we are highly motivated to investigate the potential computational benefit of porting cloud microphysics parameterization to GPU. It is also evident that there are multiple ways for GPU offload and each of them has its own pros and cons. In this study, we choose the directive-based method as the best fit for our development cycle, and below are our major justifications:

- The directive-based method is implemented by adding GPU directives to the source code and they look the same as the code comments, rarely affecting the structure and readability of the original code.
- Compared to CUDA and Kokkos, the directive-based method does not require a rewrite of the existing source code, which are frequently written in Fortran, into CUDA or C++. Rewriting the existing Fortran code in another language represents a significant burden and distraction from additional science development for this effort. Fortunately, we are not negatively impacted by the known limitations of directive-based approaches with respect to modern data structures (e.g., type-bound procedures) in this study.
- Compared to CUDA and CUDA Fortran, the directive-based method allows the same source code to run on either the CPU or GPU, instead of forcing a user to use GPU only. Since scientific code is frequently under active development, using the directive-based method improves portability and maintainability by eliminating the need to maintain multiple versions of code in different languages.
- The directive-based method allows incremental changes during code development. In particular, we are able to run some parts of a code on the GPU while others remain on the CPU. This incremental approach enables us to verify our GPU modifications step by step, allowing the early identification of a code bug. Note that translating Fortran to C++ is also considered an incremental approach and users could explore it in their own work, too.

19422466, 2023, 5, Downloaded from https://agupubs.onlinelibrary.wiley.com/doi/10.1029/2022MS003515 by Battelle Memorial Institute, Wiley Online Library on [21/11/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

Table 1
Total Number of Columns (for Physics Parameterizations) Over the Globe and Its Equivalent Resolution in a Regular Latitude-Longitude Grid Under Different Horizontal Resolutions, Using the Spectral-Element Dynamical Core

Horizontal resolution	Equivalent resolution in a regular latitude-longitude grid	Number of columns over the globe (i.e., N_c in the context)
ne30	1°	48,600
ne120	0.25°	777,600

There are two widely used directive-based methods for GPU offload: OpenACC and OpenMP target offload. We will examine both methods in this study and find out which method achieves better performance, compared to the traditional CPU-based method for our particular combination of compiler and hardware that are described in Section 2.5. Note that the resulting performance of OpenACC and OpenMP target offload is highly dependent on the choice of selected compiler and architecture.

The remainder of this paper is organized as follows. Section 2 gives a brief overview of the global climate model used in this study, the hardware and software for performance evaluation, how we apply the directive-based method to the cloud microphysics parameterization, and how we verify our GPU changes. Section 3 presents the results from the box model and global simulations and Section 4 draws the conclusion.

2. Methodology

2.1. Model Description

The global climate model used in this study is CAM version 6 (CAM6, Danabasoglu et al., 2020). It uses the Spectral-Element (SE) dynamical core (Dennis et al., 2012; Taylor & Fournier, 2010) and many physics parameterizations that are documented in details in Danabasoglu et al. (2020). CAM6 uses the Morrison-Gettelman two-moment cloud microphysics parameterization (MG2, Bacmeister et al., 2014; Gettelman & Morrison, 2015; Gettelman et al., 2020; Sun et al., 2019; Zheng et al., 2016) by default and the MG3 parameterization can be used alternatively to additionally simulate the graupel or hail (Gettelman et al., 2019). The MG parameterization has been renamed to PUMAS to reflect the broader contributions from the research community (Gettelman et al., 2023). Note that the same PUMAS code base can be configured for either the MG2 or MG3 parameterization. This study focuses on the MG3 parameterization for code porting and performance analysis because MG3 takes about 5%–10% more computational time than MG2 due to the additional processes for rimed ice (graupel or hail). The detailed list of major components in PUMAS could be found in Morrison and Gettelman (2008). The SE dynamical core in CAM6 supports a coarse horizontal resolution at ne30 (equivalent to 1° resolution in a regular latitude-longitude grid, see Table 1) and a finer resolution at ne120 (equivalent to 0.25° resolution). There are 32 vertical layers with the model top up to 2.26 hPa (about 40 km).

2.2. Generation of a PUMAS Stand-Alone Kernel With Additional Features

When porting PUMAS to GPU, we could directly modify the source code and evaluate its impact in a practical CAM simulation. However, it is time- and resource-consuming to build and run a CAM simulation. Therefore, a more computationally efficient way is to perform the GPU offload and optimizations of PUMAS in a box model or stand-alone kernel first, similar to the work of Sun et al. (2018). In this study we use Kernel GENerator (KGEN, Y. Kim et al., 2016), a Python-based open-source tool, to generate the stand-alone PUMAS kernel. By applying KGEN to a CAM simulation, KGEN will extract the desired subroutine (e.g., the main subroutine in PUMAS) specified by a user and its dependent subroutines or modules. In addition, KGEN will automatically collect the input and output data for the extracted subroutine from the CAM simulation so that the generated kernel could run and be verified against the CAM output. More details about how KGEN works as an automated kernel extraction tool could be found in Y. Kim et al. (2016). Note that the kernel generated by KGEN is a serial code no matter whether the host model is MPI-enabled or not. This serial code does not accurately represent how PUMAS will be executed in practice both on the CPU and GPU. Hence we manually add the MPI interface in order to overcome this drawback. In addition, most supercomputers have more CPU cores than GPUs within the same compute node. This means a program that uses a full node can access the same GPU from different CPU cores. Thus the added MPI interface also allows us to evaluate the impact of oversubscribing the same GPU with multiple MPI ranks on the GPU performance. For simplicity, each MPI rank will work on the same data redundantly but independently. The performance of using MPI in the PUMAS stand-alone kernel will be presented in Sections 3.1.2 and 3.1.3. The global grid is divided into N_c columns in CAM and the value of N_c varies a lot between different horizontal resolutions (see Table 1). When preparing

19422466, 2023, 5, Downloaded from https://agupubs.onlinelibrary.wiley.com/doi/10.1029/2022MS003515 by Battelle Memorial Institute, Wiley Online Library on [21/11/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

the input data for the generated kernel, KGEN will automatically select certain columns from different MPI ranks and/or OpenMP threads and from different invocations to the same subroutine. The maximum number of columns that can be extracted per MPI rank and/or OpenMP thread is 16 by default and the actual number of columns for PUMAS could be smaller, depending on whether a column contains cloud and the globe could be divided evenly by the given number of MPI ranks. If we want more columns, we need to increase the value of a configuration parameter called "PCOL" in CAM and re-run a new CAM simulation, which is not convenient. Therefore, We add a new variable α in the generated PUMAS stand-alone kernel to arbitrarily replicate the input columns by a factor of α, mimicking the partial functionality of "PCOL." This feature will not change the answers since the replicated columns have the same values as the original input. However, the α parameter allows for an easy increase of the problem size, which is useful for the CPU and GPU scaling tests shown in Section 3.1.

2.3. Verification of the GPU-Enabled PUMAS in CAM

In the section above, we describe that we use KGEN to generate a PUMAS stand-alone kernel and to initiate the GPU porting. As mentioned in Section 2.2, KGEN will collect the output data for the desired subroutine from a practical CAM simulation and the PUMAS stand-alone kernel can reproduce the same result compared to the CAM output, as long as the same compiler version, compiler flags and hardware are used. However, if we switch from one compiler to another or from CPU to GPU, the new output will likely not be bit-for-bit (BFB) identical to the original CAM output and such a difference is typically a few orders of magnitude larger than machine precision. Therefore, we are unsure whether a non-BFB change due to the GPU offload in the PUMAS stand-alone kernel is expected (e.g., inconsistent hardware and software) or not (e.g., a code bug). One way to verify the correctness of our GPU changes is to bring the revised PUMAS code back to CAM, perform a long-term (e.g., 10 years) simulation, and examine whether the climatology is changed (switching hardware is not expected to change the climate (Baker et al., 2015; Milroy et al., 2016)). However, the computational expense of this approach is huge and it also requires subjective evaluation of the climatological output, which may not be feasible for a less experienced researcher with limited resources. A more computationally efficient way to address the same challenge is the ensemble consistency test (ECT) in CAM, which is also used in many other studies (Y. Kim et al., 2016; Milroy et al., 2016; Zhang et al., 2020). ECT is designed to evaluate whether a code change would lead to a statistically different climatology by comparing the test simulation with the code change to a trusted baseline. In this study, we follow the configurations suggested by Baker et al. (2015) and Milroy et al. (2018) to set up the ECT simulations and verify that our GPU changes do not impact the correctness of CAM. Note that the most time-consuming part of ECT is generating the trusted baseline which is only performed once. A test simulation can be performed relatively quickly since it requires many fewer ensembles than the baseline. The usage of ECT allows correctness to be maintained without negatively affecting the speed of the development cycle.

2.4. PUMAS Code Refactoring for GPU

Before porting the PUMAS code to GPU, we first go through the code and realize that some code refactoring efforts are required to enable the effective usage of GPUs.

- PUMAS has many two-dimensional arrays such as qc(ncol, nlev) and they are initialized by the use of array syntax (e.g., qc = 0.0). While this is a supported Fortran syntax, unfortunately, array syntax may not work on the GPU, depending on the specific implementation. As a result, we need to explicitly specify the index of each dimension:

```
do k = 1, nlev
   do i = 1, ncol
      qc (i,k) = 0
   end do
end do
```


19422466, 2023, 5, Downloaded from https://agupubs.onlinelibrary.wiley.com/doi/10.1029/2022MS003515 by Battelle Memorial Institute, Wiley Online Library on [21/11/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

- We observe that many subroutines in PUMAS are invoked using statements like:

```
do k = 1, nlev
...
   call size_dist_param_basic(mg_snow_props, qsic(:,k), nsic(:,k), lams(:,k), ncol, n0 = n0s(:,k))
...
end do
```

- This type of call structure limits the amount of available parallelism to ncol, which is unfortunately insufficient for an efficient GPU implementation. To address this limitation, we push the vertical loop into the subroutine so that the amount of simultaneous work available for the GPU is now ncol × nlev. A revised implementation looks like the example below:

```
call size_dist_param_basic(mg_snow_props, qsic, nsic, lams, ncol, nlev, n0 = n0s)
```

- Initially a huge loop over the vertical layers in PUMAS (about 1,000 lines) unnecessarily limits the amount of parallelism available to the GPU due to a loop-carried dependency in the vertical dimension of a small number of calculations. Thus we break this huge loop into several smaller loops (fewer than 100 lines) before we start the GPU porting. While this likely increases the cost of kernel launch overhead, it also significantly reduces the register pressure as a trade-off. The net results of our loop fission, in this particular case, lead to a reduction in the execution time on the GPU.
- Some *elemental* functions/subroutines are used in PUMAS. Elemental functions are convenient in that they use the same interface for either scalar or array arguments. However, we find that their use negatively impacts performance. The reason for this is that the *elemental* procedure is called for each element of the array which prevents any form of compiler vectorization over the array elements. Therefore, we remove the *elemental* keywords and explicitly generate separate interfaces for scalar and array arguments.

We verify that all the changes above are BFB on the CPU with the same compiler and flags, and they save about 10% of the computational time of PUMAS on the CPU (the main contributor to the improved CPU performance is the removal of the *elemental* functions/subroutines). After these code refactoring efforts, we then add the OpenACC and OpenMP target offload directives to port PUMAS from CPU to GPU. A code example is presented and described shortly in Figure 2 and Figure S1 in Supporting Information S1, and more details could be found in the source code provided together with this paper.

2.5. Hardware and Software

The Casper system at the NCAR Wyoming Supercomputing Center (NWSC), which is used for this study, is a heterogeneous machine with both pure CPU nodes and CPU-GPU-mixed nodes (Computational and Information Systems Laboratory, 2019). All the compute nodes used in this study have two sockets and each socket has 18 CPU cores (2.3-GHz Intel Xeon Gold 6140 (Skylake)). Among the total 10 compute nodes with GPU available, four of them have four NVIDIA Tesla V100 GPUs per node and the remaining six have eight NVIDIA Tesla V100 GPUs per node. Note that Casper is not designed to perform production-level GPU computing, so a maximum of four GPU nodes can be requested by a single user per job. Therefore, we will only use a limited number of GPU nodes on Casper for performance evaluation in this study. Both Intel and NVIDIA High-Performance Computing (NVHPC) compilers are installed on Casper. The NVHPC compiler supports both OpenACC version 2.6 and OpenMP version 5.1 as directive-based methods for GPU offload. In addition, the NVIDIA Nsight Systems and Nsight Compute are available as profiling tools to identify the performance bottlenecks and suggest potential optimizations. In this study, we use Intel/19.1.1 compiler for the CPU and NVHPC/22.2 compiler for the GPU. While multiple CPU compilers exist on Casper, the Intel/19.1.1 compiler consistently generates the executable that minimizes the execution time.

19422466, 2023, 5, Downloaded from https://agupubs.onlinelibrary.wiley.com/doi/10.1029/2022MS003515 by Battelle Memorial Institute, Wiley Online Library on [21/11/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

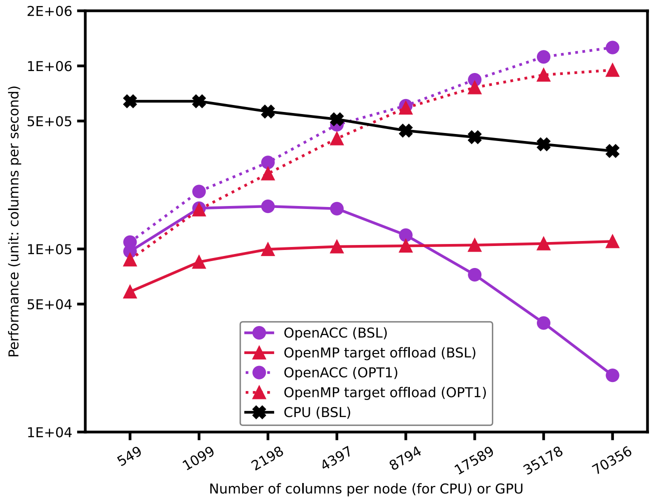

Figure 1. The CPU (one full compute node, 36 MPI ranks) and GPU (one MPI rank and one V100 GPU) performance (y-axis: log scale) of the PUMAS stand-alone kernel on Casper for the different number of columns per node or GPU (x-axis: log scale). The solid lines refer to the baseline (denoted as BSL) and the dotted lines refer to the GPU performance with the revised implementation of loops containing data dependency (denoted as OPT1).

3. Results

3.1. Performance of the PUMAS Stand-Alone Kernel

After porting the PUMAS code to GPU using OpenACC and OpenMP target offload, and verifying the results using ECT in CAM (see details in Section 2.3), it is now possible to present the performance of the PUMAS stand-alone kernel on the CPU and GPU. We will (a) demonstrate the impact of GPU computing on the performance, (b) compare the performance of OpenACC and OpenMP target offload implementations, and (c) describe GPU optimizations for further performance improvements. Because KGEN collects input data from different MPI ranks/or OpenMP threads, the number of columns per data set can vary. We, therefore, decide to use the number of columns per second as our metric to measure the performance of the PUMAS stand-alone kernel on CPU and GPU. The metric is calculated as below:

$$\text{Rate} = \frac{\alpha \cdot N_{\text{MPI}} \cdot \sum_{i=1}^{L} N_{\text{col},i}}{\frac{\sum_{k=1}^{K} \sum_{i=1}^{L} t_{i,k}}{K}} \tag{1}$$

where L is the number of input data sets collected by KGEN; $N_{\text{col},i}$ is the number of columns in the ith input data; $t_{i,k}$ is the maximum computational time per MPI rank (unit: second) for the ith input data during the kth execution; α is the factor to duplicate the column numbers (see details in Section 2.2); N_{MPI} is the number of MPI ranks per node used in a CPU or GPU run; K is the number of repeated executions. We find that $K = 500$ is sufficient to minimize any impact that the timer granularity has on our measurements. Note that the performance metric here excludes the data movement time between CPU and GPU, and only accounts for the creation of temporary variables and computation on GPU. This is done by explicitly performing the data movement between CPU and GPU before invoking the PUMAS subroutines so that there is no data transfer during the computations on the GPU. The CPU performance metric is always collected for a full compute node (i.e., 36 CPU cores or 36 MPI ranks per node). The GPU performance metric is measured for the three specific configurations below within the same node and the results will be shown in the following sections individually:

- One MPI rank and one GPU
- Multiple MPI ranks and one GPU
- Multiple MPI ranks and multiple GPUs

When comparing the CPU and GPU performance, we mostly use the following two metrics for evaluation:

$$\text{Speedup} = \frac{\text{Rate}_{\text{GPU}}}{\text{Rate}_{\text{CPU}}}, \text{ if GPU is faster than CPU} \tag{2}$$

$$\text{Slowdown} = \frac{\text{Rate}_{\text{CPU}} - \text{Rate}_{\text{GPU}}}{\text{Rate}_{\text{CPU}}} \times 100\%, \text{ if GPU is slower than CPU} \tag{3}$$

Table 2
List of the Experiments Performed in Section 3.1.1 With the PUMAS Standard-Alone Kernel

Case	Details
BSL	The baseline or initial results
OPT1	Revised implementation of some loops containing data dependency
OPT2	OPT1 + revised implementation for parallelized sedimentation calculations

19422466, 2023, 5, Downloaded from https://agupubs.onlinelibrary.wiley.com/doi/10.1029/2022MS003515 by Battelle Memorial Institute, Wiley Online Library on [21/11/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

```
!$acc parallel
!$acc loop seq
!$omp target teams
!$omp loop bind(teams)
    do k = 1, nlev
        do i = 1, mgncol
            a (i,k) = a(i,k-1) + b(i,k)
        end do
    end do
!$acc end parallel
!$omp end target teams
```

```
!$acc parallel
!$acc loop gang vector
!$omp target teams
!$omp loop bind(teams)
    do i = 1, mgncol
        !$acc loop seq
        !$omp loop bind(teams)
        do k = 1, nlev
            a (i,k) = a(i,k-1) + b(i,k)
        end do
    end do
!$acc end parallel
!$omp end target teams
```

Figure 2. The baseline code structure (referred to BSL in Table 2, left panel) and the optimized code structure for GPU (referred to OPT1 in Table 2, right panel).

3.1.1. One CPU Node Versus One MPI Rank and One GPU

The initial CPU (36 MPI ranks) and GPU (one MPI rank and one GPU) performance of the PUMAS stand-alone kernel are plotted as the solid lines in Figure 1. These performance results serve as the baseline for both CPU and GPU runs (denoted as BSL in Figure 1 and Table 2). We vary the duplication factor α to specify different numbers of columns per node (for CPU runs) or per GPU (for GPU runs). In particular, we begin with 549 columns to reflect the value when we use 36 MPI ranks for a practical CAM simulation. Based on the initial results, the CPU performance degrades along with the increased number of columns. This is reasonable as more columns mean more data and they may no longer fit the CPU cache size at some point. Unless we chunk the data explicitly, this will lead to more data movement between cache and memory and slow down the performance. In contrast, the GPU performance is worse than the CPU performance for all the runs and we observe different behaviors between OpenACC and OpenMP target offload. OpenACC outperforms OpenMP target offload when the number of columns per GPU increases from a small value. However, the performance of OpenACC drops quickly when the number of columns becomes large and it is even worse than the performance of OpenMP target offload. On average, using OpenACC and OpenMP target offload for GPU runs slows down the performance by about 79%, compared to the CPU performance.

The behavior of GPU performance in the baseline is unexpected. Therefore we use the NVIDIA Nsight Systems tool to profile the PUMAS stand-alone kernel and try to identify the source of the bottleneck. The profiling results reveal that the major penalty of GPU performance comes from approximately 10 loops with data dependencies, especially when the number of columns is large. A problematic data dependency is illustrated in the left panel of Figure 2. While this code structure optimizes the memory access patterns for the column-major Fortran order, due to the dependency between $a(i, k)$ and $a(i, k-1)$, the outer loop has to be executed serially to ensure the correctness. A straightforward port to GPU results in the serial execution of both loops, leading to poor usage of GPU resources. In the right panel of Figure 2, we switch the loop order. This loop transformation enables GPU parallelization of the outer loop at the cost of uncoalesced memory access due to the lack of unit stride for the inner loop. The updated GPU performance of the PUMAS stand-alone kernel with the changes of loop order is plotted as the dashed lines in Figure 1 (denoted as OPT1). The GPU performance is still worse compared to the CPU performance when the number of columns is small. However, the GPU performance improves rapidly with a larger number of columns and it starts to outperform the CPU performance when there are greater than 4,400 columns per node or GPU. Compared to the CPU performance on Casper, the GPU performance (OpenMP target offload) can be 86% slower when there are only 549 columns per GPU but achieves 3.7× speedup (OpenACC) when there are 70,356 columns per GPU. The updated GPU

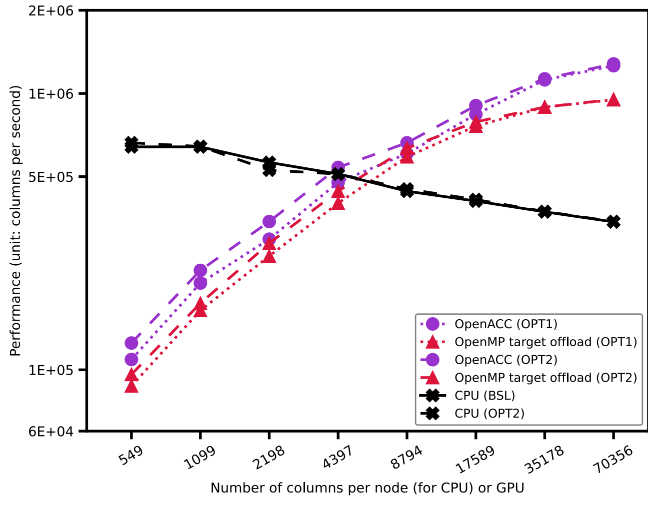

Figure 3. Similar to Figure 1 but remove the BSL results for GPU. The dashed lines represent the CPU and GPU performance with the revised implementation of some loops containing data dependency and parallelized sedimentation calculations (denoted as OPT2).

19422466, 2023, 5, Downloaded from https://agupubs.onlinelibrary.wiley.com/doi/10.1029/2022MS003515 by Battelle Memorial Institute, Wiley Online Library on [21/11/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

performance agrees better with our expectations since GPU has higher computation capacity than CPU and prefers to work on more data at once. In addition, OpenACC is about 1.2× as fast as OpenMP target offload on average.

We further look into the new profiling results for the GPU performance and find out that the most time-consuming part in the PUMAS stand-alone kernel now is the sedimentation process for five hydrometeors (cloud water, cloud ice, rain, snow, and graupel). This observation is consistent for both OpenACC and OpenMP target offload implementations, and from a small to a large number of columns. The original sedimentation for each hydrometeor is calculated sequentially but it could be modified to enable a concurrent execution on GPU for all hydrometeors. This revision further improves the OpenACC and OpenMP target offload performance by another 1.1× speedup, as shown in Figure 3 (denoted as OPT2). To understand the enhanced GPU performance here better, we use the Nsight Systems tool to profile the revised GPU code and it first confirms that the sedimentation calculation for different hydrometeors is executed asynchronously as expected. When there are 549 columns per GPU, the kernel execution time of sedimentation ranges from 120 to 840 μs, while the corresponding kernel launch latency ranges from seven to 930 μs. When the number of columns per GPU increases to 70,356, the kernel execution time ranges from 694 μs to 7.56 ms, while the corresponding kernel launch latency ranges from 1.26 to 12.71 ms. It seems that the kernel launch latency is high for the sedimentation calculation and would adversely affect the GPU performance at the first glance. However, the long kernel launch latency of a particular sedimentation calculation is well hidden by the execution of other sedimentation calculations and thus does not degrade the GPU performance in this case. We further use the Nsight Compute tool to perform a Roofline model analysis of the most expensive sedimentation kernel. The Roofline model analysis allows us to evaluate the GPU performance of a given kernel against the hardware capabilities (e.g., peak performance and memory bandwidth) and identify the bottleneck or limitation of that kernel during the software development cycle. The Roofline model results (Figure S2 in Supporting Information S1) first identify that the sedimentation kernel falls into the bandwidth-bound regime, which means that saturating the memory bandwidth is more critical to achieving good performance on the GPU. When there are only 549 columns per GPU, the sedimentation kernel is far away from the peak memory bandwidth, which means a poor memory throughput on the GPU. When the number increases to 70,356 columns per GPU, the same sedimentation kernel is much closer to the peak memory bandwidth and thus attains a higher memory throughput on the GPU compared to the former case. High memory throughput on the GPU is able to mitigate the impact of kernel launch latency on the GPU performance, which then explains the improved GPU performance with an increased number of columns per GPU as observed in Figure 3. We also examine additional optimization strategies like executing more subroutines asynchronously, further breaking some GPU kernels into smaller ones, and changing the vector length for OpenACC. Because these changes have a smaller impact relative to the previously mentioned modifications, we then do not explicitly report them here. On the other hand, since our goal is to maintain a single source code for both CPU and GPU execution, we re-measure the CPU performance with the GPU changes described above. The updated CPU results are provided in Figure 3. Fortunately, the CPU performance is minimally impacted, which demonstrates that we can maintain the CPU performance while obtaining better performance on GPU. In addition, OpenACC is about 1.2× as fast as OpenMP target offload with the optimizations above, suggesting that OpenACC is a more appropriate choice for PUMAS at this time.

3.1.2. One CPU Node Versus Multiple MPI Ranks and One GPU

In the previous section, we focus on the configuration of one MPI rank and one GPU for GPU performance. In reality, there are typically more CPU cores than GPUs per node. Thus we would naturally raise a question: shall we use multiple CPU cores to drive a single GPU? In addition, CAM is designed to use consistent MPI ranks for all the atmospheric components. This means that if we use one MPI rank for PUMAS, we have to use one MPI rank for the other physics parameterizations. Since most physics parameterizations run exclusively on CPU, using one MPI rank for them will lead to a significant waste of computational resources and a degraded performance. Therefore, in order to avoid sacrificing the performance of other CAM components, we have to oversubscribe the same GPU with multiple MPI ranks and we would like to demonstrate how this configuration affects the GPU performance in this section. Note that NVIDIA provides the Multi-Process Service (MPS) for its GPU products, which allows multiple MPI ranks to utilize a single GPU simultaneously as long as there is enough resource. Note that for all the tests below when driving a given GPU with multiple MPI ranks, the total number of columns per GPU shown in the figures is the sum of the columns from its associated MPI ranks.

19422466, 2023, 5, Downloaded from https://agupubs.onlinelibrary.wiley.com/doi/10.1029/2022MS003515 by Battelle Memorial Institute, Wiley Online Library on [21/11/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

Figure 4. The GPU performance (y-axis: log scale) of OpenACC and OpenMP target offload with and without Multi-Process Service (MPS) for different numbers of columns per GPU (x-axis). All the tests use two MPI ranks and one GPU.

To illustrate its impact, we use OpenACC and OpenMP target offload with and without MPS and the GPU performance (two MPI ranks and one GPU) is summarized in Figure 4. It turns out that when there is a small number of columns per GPU, turning on MPS leads to a remarkably improved performance for both OpenACC and OpenMP target offload since it directly increases the number of columns that can be calculated on GPU and is able to overlap the kernel execution time and kernel launch latency between two MPI ranks. When we increase the number of columns, turning on MPS is still beneficial but its impact on the performance is much smaller. This is caused by the fact that there are not enough resources to allow two MPI ranks to run completely in parallel on the same GPU. On the other hand, we have confirmed that turning on MPS does not negatively affect the GPU performance of the configuration examined in Section 3.1.1 (e.g., using one MPI rank and one GPU). Therefore, based on the results here, we will always turn on MPS in the following tests. It is also evident that OpenACC still outperforms OpenMP target offload (1.2× speedup with MPS and 1.1× speedup without MPS) in this configuration, consistent with the results in Section 3.1.1. Therefore, we will exclusively focus on OpenACC for the remaining evaluations of GPU performance.

Next, we vary the number of MPI ranks from one to 36 MPI ranks per GPU and provide the OpenACC GPU performance in Figure 5. We observe that when there is only a small number of columns, the GPU performance degrades if we assign more than one MPI rank to the same GPU. In addition, the more MPI ranks launched per GPU, the worse GPU performance is (up to 86% slower when oversubscribing the same GPU with 36 MPI ranks). The difference in GPU performance becomes smaller when we increase the number of columns and assigning multiple MPI ranks to the same GPU may perform slightly better when there is a large number of columns. On the other hand, the GPU performance of oversubscribing the same GPU with multiple MPI ranks can still outperform the CPU performance. However, more columns are needed to be calculated on the GPU simultaneously in order to achieve better performance. Overall, we do not observe a clear computational benefit while oversubscribing the same GPU with multiple MPI ranks, unless there is a large number of computations done on the GPU. According to the profiling results, given the same number of columns per node, using more MPI ranks leads to fewer threadblocks (in CUDA's terminology) per MPI rank. Fewer threadblocks will fail to hide the latency and reduce the throughput on the GPU, hurting the overall performance. In addition, it takes time for the CUDA driver to switch between different MPI ranks and get the computational resources ready for each MPI rank. This time is generally modest but will become non-negligible when the execution time of a compute kernel is small, such as the situation with a small number of columns per node here. Therefore, the findings here indicate that in general, we should assign only one MPI rank to one GPU. For a real application like CAM, this means that we should port more computations

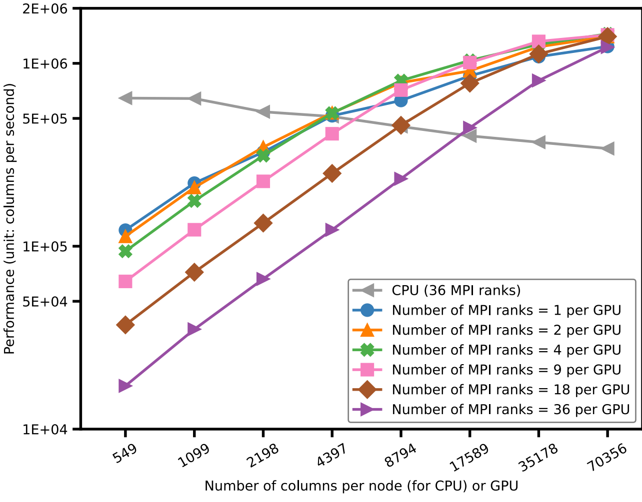

Figure 5. The OpenACC GPU performance (y-axis: log scale) with multiple MPI ranks and one GPU for different numbers of columns per node or GPU (x-axis: log scale). MPS is turned on for all the GPU runs. The CPU performance is obtained from Figure 3 for comparison.

19422466, 2023, 5, Downloaded from https://agupubs.onlinelibrary.wiley.com/doi/10.1029/2022MS003515 by Battelle Memorial Institute, Wiley Online Library on [21/11/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

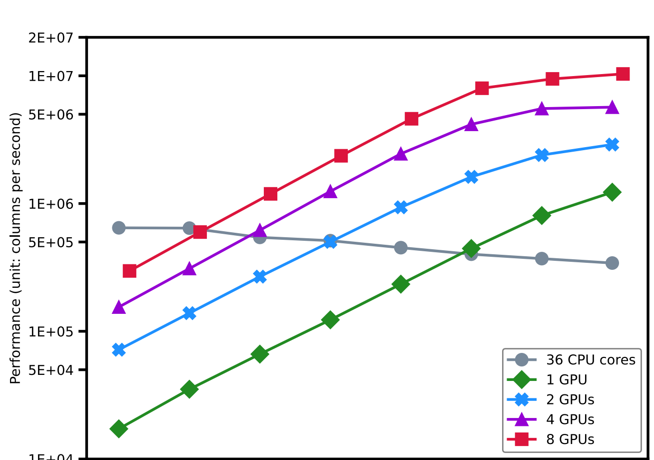

Figure 6. The OpenACC GPU performance (y-axis: log scale) for different numbers of columns per node (x-axis: log scale) from weak-scaling tests. Each test uses 36 MPI ranks and up to eight GPUs within the same node. The CPU performance from Figure 5 is also added for comparison.

besides PUMAS to GPU. Otherwise, since most components in CAM are still running on the CPU only, using one MPI rank for a CAM simulation will lead to poor performance even if the GPU-enabled PUMAS may benefit from it. In contrast, oversubscribing the same GPU with multiple MPI ranks can be a temporary solution when we are porting CAM components to GPU gradually and have to use 36 MPI ranks per node on Casper at this moment. It is worth noting that assigning 36 MPI ranks to a single GPU starts to outperform a full CPU node (36 CPU cores) when the number of columns per node is more than 18,000. This value is slightly larger than that for a CAM simulation at 2° resolution but much smaller than that for a CAM simulation at a 1° resolution (see Table 1 and assume using a single compute node). Therefore, we expect that PUMAS can benefit from GPU computing in a practical CAM simulation as well.

3.1.3. One CPU Node Versus Multiple MPI Ranks and Multiple GPUs

In the previous section, we presented that oversubscribing the same GPU with multiple MPI ranks can degrade the GPU performance, compared to assigning one MPI rank to one GPU. However, this performance degradation could be mitigated if fewer MPI ranks are launched on the same GPU. In reality, given the fixed number of MPI ranks, this is attained by exploiting the multiple GPUs within the same node. Figure 6 shows the GPU performance with 36 MPI ranks and up to eight GPUs per node. These results come from the weak-scaling tests, meaning that the number of columns per GPU is the same and thus using more GPUs leads to a larger total number of columns per node. Note that 36 MPI ranks can not be distributed across eight GPUs evenly. Thus four GPUs will work on more columns than the other four and we select the maximum number of columns per GPU as the x-axis value here, leading to a slight shift of the performance curve of eight GPUs in Figure 6. It turns out that compared to using one GPU per node, using two, four, and eight GPUs per node achieves 3.5×, 5.8×, and 8.9× speedup on average, respectively. The computations on different GPUs are executed concurrently and independently, but the speedup here is larger than the difference between the used GPUs. This is because using more GPUs per node leads to fewer MPI ranks per GPU, improving the GPU performance according to Figure 5. We also perform a weak-scaling test using eight MPI ranks and eight GPUs, and the results are shown in Supporting Information S1 (Figure S3). It achieves close to 8× speedup as expected, compared to using one MPI rank and one GPU. It also outperforms the 36 CPU cores even with 549 columns per GPU, which again exhibits the advantage of assigning one MPI rank to one GPU. Overall, based on the results from the box model, we have demonstrated that porting PUMAS to GPU is beneficial as long as the computational burden is heavy enough.

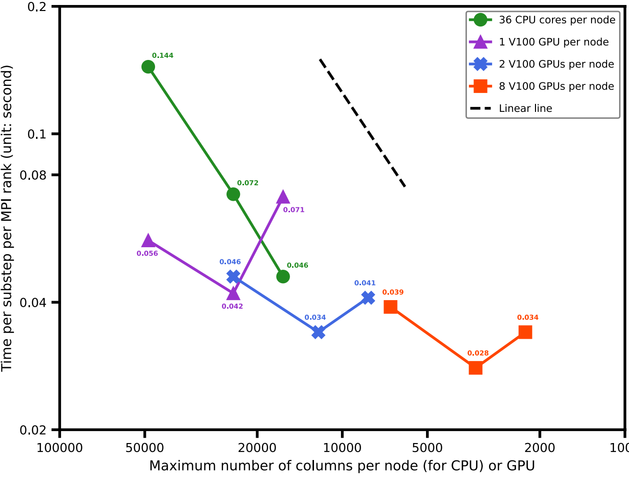

Figure 7. The CPU and GPU performance (y-axis: log scale) of PUMAS in a 1-day CAM simulation for a strong-scaling test (x-axis: log scale). The CAM simulation is performed with the spectral-element dynamical core at ne30 resolution and up to three compute nodes on Casper. The black dashed line is added as a reference for linear scalability.

3.2. Performance of the GPU-Enabled PUMAS in CAM

In this section, we bring the GPU-enabled PUMAS code described in Section 3.1 into CAM (with a few additional scientific updates) and evaluate its GPU performance in a practical simulation. The results are grouped into two subsections: (a) the "Base case" subsection where a given GPU is oversubscribed by multiple MPI ranks, and (b) the "Optimal case" subsection where only one MPI rank is assigned to one GPU.

3.2.1. Base Case

The performance here is collected from the simulations using the ne30 horizontal resolution. We vary the number of compute nodes to perform a strong-scaling test, meaning that the problem size is fixed, and using more compute nodes results in fewer computations per node or GPU. Note that the GPU performance here contains the time of (a) data copy from CPU to GPU, (b) allocation of temporary data on the GPU, (c) computation on the GPU, and (d) data copy from GPU to CPU. We would like to examine their individual

19422466, 2023, 5, Downloaded from https://agupubs.onlinelibrary.wiley.com/doi/10.1029/2022MS003515 by Battelle Memorial Institute, Wiley Online Library on [21/11/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

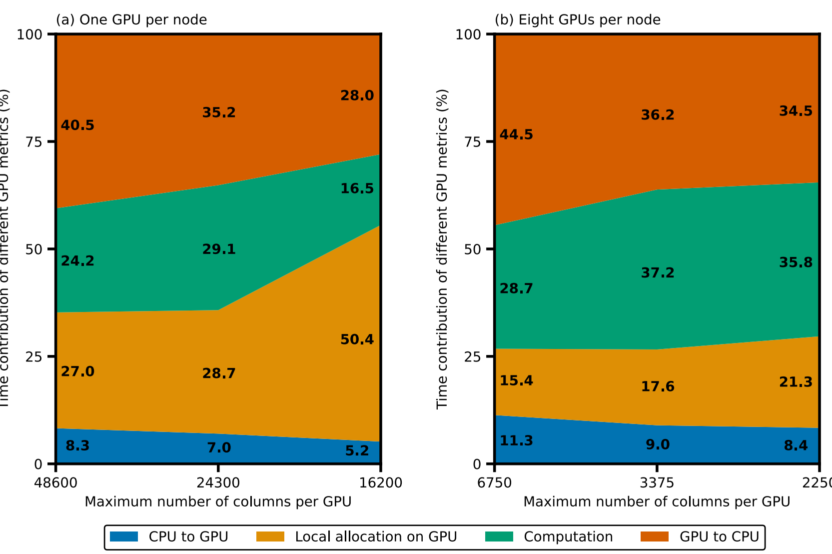

Figure 8. The time contributions (in percent) of (1) data copy from CPU to GPU, (2) allocation of temporary data on the GPU, (3) computation on the GPU, and (4) data copy from GPU to CPU to the GPU performance (use one (left panel) and eight (right panel) GPUs per node) as shown in Figure 7.

impacts in a CAM simulation with different configurations. The simulation length is 1 day and the averaged maximum wall-clock time per MPI rank is reported as the CPU and GPU performance metrics. Since PUMAS is sub-cycled per time step in CAM, the performance metric is further averaged for each substep. For each simulation, we maximize the number of columns that could be offloaded to the GPUs per substep to prevent frequent data movement between CPU and GPU.

The CPU and GPU performance of PUMAS in CAM is depicted in Figure 7. Up to three compute nodes are used for the strong-scaling tests, and each node uses 36 MPI ranks for both CPU and GPU simulations. Regarding the GPU simulations, we vary the number of GPUs per node from one to eight. It turns out that when using a single compute node (referring to the leftmost point of each curve), one GPU is able to outperform 36 CPU cores and achieve 2.6× speedup. If we focus on the GPU simulations with multiple GPUs per node, using two or eight GPUs per node can further outperform the performance of using one GPU per node by a factor of 1.2 and 1.4, respectively (equivalent to 3.1× and 3.6× speedup compared to the CPU performance). This is consistent with the observation in Figure 6, where the performance of using eight GPUs per node with 6,750 columns is slightly better than that of using two GPUs per node with 24,300 columns per node. In addition, the CPU execution time is reduced almost linearly when using multiple compute nodes, revealing a close to linear scalability for a strong-scaling test as expected. In contrast, the GPU performance in a two-node run (referring to the middle point of each curve) only obtains about 1.3× to 1.4× speedup compared to that in a one-node run, and could even degrade in a three-node run (referring to the rightmost point of each curve). This suggests that though the GPU performance could still outperform the CPU performance with increased number of nodes, the GPU runs do not scale as well as the CPU runs. To understand this behavior, we further investigate the time contributions of data movement and computation to the overall GPU performance and the results for different nodes are shown in Figure 8. The analyses for the simulations with one and eight GPUs per node are presented here for comparison. When the number of used nodes increases, the maximum number of columns per GPU decreases consequently as shown in the *x*-axis. It turns out that the data movement and allocation on the GPU are more time-consuming than the computation itself throughout all the simulations. In particular, the contribution of data movement and allocation time on the GPU is reduced with two nodes but increases with three nodes. Further profiling results indicate that when using one GPU per node, the GPU kernels are overlapped more in the three-node configuration, which indicates a stronger competition of GPU resources compared to the one-node configuration. Hence the data movement, allocation, and computational time on GPU increases accordingly, especially for the data allocation on the GPU. This then explains the difference in the multi-node GPU performance observed in Figure 7. Similar behavior is observed for the simulations with eight GPUs per node. However, fewer GPU kernels are overlapped and the data movement and allocation time on the GPU also decreases. Therefore, it is reasonable to obtain a better performance and scalability by exploiting multiple

19422466, 2023, 5, Downloaded from https://agupubs.onlinelibrary.wiley.com/doi/10.1029/2022MS003515 by Battelle Memorial Institute, Wiley Online Library on [21/11/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

19422466, 2023, 5, Downloaded from https://agupubs.onlinelibrary.wiley.com/doi/10.1029/2022MS003515 by Battelle Memorial Institute, Wiley Online Library on [21/11/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

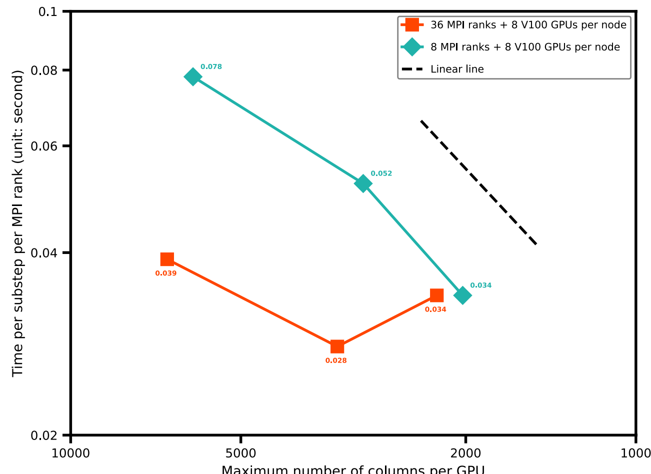

Figure 9. The GPU performance (y-axis: log scale) of PUMAS in a 1-day CAM simulation for a strong-scaling test (x-axis: log scale), with (red line with square markers) and without (green line with diamond markers) oversubscribing the same GPU with multiple MPI ranks. The CAM simulation is performed with the spectral-element dynamical core at ne30 resolution and up to three compute nodes on Casper. The black dashed line is added as a reference for linear scalability.

GPUs per node, compared to the configuration with one GPU per node as shown in Figure 7. It is worth noting that the time of data transfer from GPU to CPU is generally about four to five times that from CPU to GPU, which is caused by the fact that there are 37 input variables but 146 output variables in PUMAS. Since PUMAS is the only component running on the GPU in our simulations, we have to initiate those data transfers every time the PUMAS calculations are invoked. When the whole CAM is ported to GPU, we could make the input and output data resident on the GPU during the entire simulation. Thus the speedup can be enhanced further by eliminating unnecessary data transfer.

3.2.2. Optimal Case

In Section 3.2.1, the GPU performance of PUMAS in CAM is presented with the configuration of oversubscribing the same GPU with multiple MPI ranks. The underlying reason is explained in Section 3.1.2 but based on the results in that section, we also know that oversubscribing the same GPU with multiple MPI ranks could hurt the GPU performance in general. Therefore, we would like to perform a sensitivity test, which adopts the same configurations in Section 3.2.1 (denoted as CFG1) except that we now use only eight CPU cores and eight GPUs per node (denoted as CFG2). This allows us to keep just one MPI rank per GPU and thus avoid the occurrence of oversubscription. We understand that by doing this, we will slow down the performance of other CAM components significantly since they run exclusively on the CPU and we do not utilize the whole CPU cores per node. Nevertheless, the purpose of the sensitivity test here is to examine whether this configuration is also optimal for the GPU performance of PUMAS in CAM, compared to that in Section 3.2.1. The GPU performance of CFG2 is depicted in Figure 9 and the GPU performance of CFG1 in Figure 7 is added for comparison. It turns out that CFG2 leads to a much better scalability than CFG1. This is attributed to the fact that there is no competition for the same GPU resources between different MPI ranks in CFG2. The observation here also confirms our analysis about the poor scalability of CFG1 in Section 3.2.1. It is worth noting that though the scalability of CFG2 is clearly improved, the actual GPU performance is worse, especially when only one compute node is used. This is caused by the fact that CFG2 uses eight MPI ranks per node, with 6,075 (=48,600 ÷ 8) columns per MPI rank in a one-node run. It is much higher than the number from CFG1, which is 1,350 (=48,600 ÷ 36). Therefore, the larger amount of data transfer between CPU and GPU, allocation of temporary data, and computation on the GPU is observed per MPI rank in CFG2, resulting in more execution time. Nevertheless, due to the better scalability, the GPU performance of CFG2 is able to match that of CFG1 quickly when running on three compute nodes. Since tens or hundreds of compute nodes are typically used in a CAM production run, we expect that CFG2 will eventually outperform CFG1 with the further increased number of compute nodes. Hence, the sensitivity test here indicates that assigning one MPI rank per GPU is also an optimal configuration for a GPU-enabled CAM simulation, consistent with what we conclude from the results of the PUMAS standard-alone kernel in Section 3.1. This further encourages us to port more CAM components to GPU so that we could adopt this optimal GPU configuration without compromising the performance on the CPU for those that are not GPU-enabled.

3.3. Sensitivity Tests

In the previous section, we evaluated the performance of GPU-enabled PUMAS in a practical CAM simulation and demonstrated that GPU could outperform CPU noticeably in a strong-scaling test. However, PUMAS and CAM can be configured in many ways for scientific purposes. Hence in this section, we will show two configurations that have the largest impact on the PUMAS performance based on our tests and investigate how they affect the CPU and GPU performance, respectively. Note that these two configurations here will lead to a different climatology than that generated by the configuration used in Section 3.2. Therefore, whether using those configurations in a particular research should not only rely on the performance presented below, but also the specific scientific goals.

19422466, 2023, 5, Downloaded from https://agupubs.onlinelibrary.wiley.com/doi/10.1029/2022MS003515 by Battelle Memorial Institute, Wiley Online Library on [21/11/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

Table 3
The CPU and GPU Performance of PUMAS in a One-Day CAM Simulation, Using the Explicit or Implicit Calculation of Sedimentation Process

Name	Calculation of sedimentation	Time per substep per MPI rank (unit: second)	Speedup
36 CPU cores (48,600 columns per node)	Explicit	0.144	–
	Implicit	0.142	1.0×
One V100 GPU (48,600 columns per GPU)	Explicit	0.014	–
	Implicit	0.011	1.3×
Two V100 GPUs (24,300 columns per GPU)	Explicit	0.014	–
	Implicit	0.010	1.4×
Eight V100 GPUs (maximum 6,750 columns per GPU)	Explicit	0.011	–
	Implicit	0.009	1.2×

Note. The CAM simulation is performed with the spectral-element dynamical core at ne30 resolution and on a single compute node on Casper. The timing information of GPU performance excludes the contributions from the data movement between CPU and GPU and the data allocation on the GPU.

3.3.1. Explicit Versus Implicit Sedimentation Scheme

As mentioned in Section 3.1.1, the calculation of the sedimentation process is the most time-consuming part of PUMAS. The default method used for sedimentation in PUMAS is an explicit method and it contains sub-steppings to ensure numerical stability. There is another option to calculate the sedimentation process implicitly, which does not require sub-steppings to maintain the numerical stability (Gettelman et al., 2023). It was originally derived from the Geophysical Fluid Dynamics Laboratory (GFDL) microphysics (Harris et al., 2020; Zhou et al., 2019) and later improved for better accuracy by Guo et al. (2021). This implicit method is brought into PUMAS and we perform a series of CAM simulations to evaluate its impact on the overall CPU and GPU performance. The results with a single compute node and multiple GPUs per node are summarized in Table 3. Only the computational time on the GPU is presented because the same amount of input, output, and local variables are used for the simulations. While the implicit method has the same execution time on the CPU compared to the explicit method, it results to a 1.2×–1.4× speedup on the GPU. There is a loop-carried dependency along the vertical dimension in the implicit method, but not in the explicit method. However, the explicit method requires sub-steppings and they are performed inside a single GPU kernel. Due to the need of running each sub-step sequentially, the explicit method fails to achieve parallelism on the GPU even if the calculations are vertically independent. In contrast, the implicit method does not require any sub-steppings and we are still able to compute the quantities that are vertically independent in parallel on the GPU. This is the main reason why we can observe a speedup by using the implicit method even though the implicit method has more kernel launches than the explicit method. The result here highlights that in addition to the modifications described in Section 3.1.1 that were guided by the profiling results, it is also critical to thinking about whether a chosen algorithm is appropriate for GPU computing. An ideal algorithm that runs efficiently on the GPU should be able to achieve massive parallelism for its detailed calculation. In a climate model, this usually means that the calculation is grid-independent, and no iteration such as the sub-stepping here is involved since most iterations have to be executed in serial, and thus a GPU can not be fully saturated. It is also noticeable that the GPUs could achieve a speedup between 10.3× and 15.8× in terms of computation alone compared to the CPU in a one-node simulation (see Table 3). This again reflects the dominant impact of data movement between CPU and GPU on the overall GPU performance as discussed in Section 3.2. It also means that putting more physics parameterizations on the GPU to reduce the data transfer will have large performance benefits.

3.3.2. High Horizontal Resolution

All the CPU and GPU simulations presented so far focus on the coarse horizontal resolution (i.e., around 1°). The SE dynamical core in CAM also supports the ne120 resolution, which is equivalent to the 0.25° resolution and serves as a high-resolution configuration. Recall the results from the previous sections, which demonstrate the advantage of the GPU is most pronounced for a larger problem size (e.g., more columns per node or per GPU). We next examine the impact that GPU enablement has on the high resolution and compare it to the existing speedup obtained on the coarse resolution in Section 3.2. Note that the ne120 resolution requires a large amount of memory and unfortunately can not fit into a single compute node. It is necessary to utilize three compute

19422466, 2023, 5, Downloaded from https://agupubs.onlinelibrary.wiley.com/doi/10.1029/2022MS003515 by Battelle Memorial Institute, Wiley Online Library on [21/11/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

Table 4
The CPU and GPU Performance of PUMAS in a One-Day CAM Simulation, Using the Spectral-Element (SE) Dynamical Core at ne120 Resolution and Three Compute Nodes on Casper

Name	Metrics	Time per substep per MPI rank (unit: second)	Speedup
36 CPU cores per node (259,200 columns per node)	Computation	0.876	–
Two V100 GPUs per node (129,600 columns per GPU)	Total time	0.185	4.7×
	Data transfer from CPU to GPU	0.024	–
	Allocation of local data on GPU	0.018	–
	Computation	0.015	–
	Data transfer from GPU to CPU	0.128	–
Eight V100 GPUs per node (maximum 36,000 columns per GPU)	Total time	0.161	5.4×
	Data transfer from CPU to GPU	0.027	–
	Allocation of local data on GPU	0.011	–
	Computation	0.013	–
	Data transfer from GPU to CPU	0.110	–

nodes with 700 GB CPU memory per node in order to successfully execute the high-resolution configurations on the CPU and GPU. Recalling Table 1, there are a total of 777,600 columns over the globe for the ne120 resolution. We maximize the number of columns on each GPU to achieve the minimum data transfer cost. Due to the per GPU memory limitation of 32 GB, we must use a minimum of two GPUs per node. The results for two and eight GPUs per node are summarized in Table 4. Interestingly, the speedup of GPU performance over CPU performance grows to a maximum of 5.4× at ne120 resolution, compared to the 1.4× at ne30 resolution with three nodes (see Figure 7). This is consistent with the findings in Figure 5 that the advantage of GPU versus CPU is greatest with a larger number of columns per node for a fixed number of MPI ranks. However, comparing the two GPU runs to one another, the speedup from two GPUs per node to eight GPUs per node is similar whether using the total time or the computational time only. Detailed profiling results show that in a practical CAM simulation, different MPI ranks on the same GPU generally launch the compute kernels at different times, unlike the PUMAS stand-alone kernel where the compute kernels launched by different MPI ranks on the same GPU are often overlapped. This difference is likely caused by the load imbalance issue in CAM, where different MPI ranks are not assigned with even workloads and they do not complete the previous calculations synchronously before the beginning of PUMAS computation. Hence there is less competition on the GPU resources in a practical CAM simulation, leading to a relatively smaller performance difference when using multiple GPUs per node. Comparing the result in this section with that in Section 3.1.3 for the PUMAS stand-alone kernel, we have observed a good agreement about some general behaviors such as GPU outperforms CPU with a large number of columns per node and using multiple GPUs per node is beneficial. It is worth noting that the tests in this section are limited to three compute nodes, leading to a much higher per-node workload compared to that in Section 3.2.1. Due to the requirement of fast throughput plus smaller time step and/or increased stiffness, many more compute nodes are needed to perform a high-resolution simulation in practice and the per-node workload could be even smaller than that in a low-resolution simulation under some situations.

4. Conclusion

Cloud microphysics plays a key role in the atmospheric component of Earth system models and is typically computationally costly. In this study, we have described the GPU porting of a cloud microphysics parameterization in CAM (i.e., PUMAS) through the use of directive-based methods (e.g., OpenACC and OpenMP target offload). To the authors' knowledge, this is the first time that a physics component is run on the GPUs in a practical CAM simulation. The directive-based approach allows us to use a single version of source code that could utilize either CPU or GPU. The resulting code is both maintainable and portable. We perform the GPU porting initially in a PUMAS stand-alone kernel generated by KGEN, which simplified code development and debugging. It turns out that with some optimizations, using even a single NVIDIA V100 GPU is able to outperform 36 Intel Skylake CPU cores in terms of computation alone when the number of columns per node (for CPU) or

19422466, 2023, 5, Downloaded from https://agupubs.onlinelibrary.wiley.com/doi/10.1029/2022MS003515 by Battelle Memorial Institute, Wiley Online Library on [21/11/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

GPU is greater than 4,400 (4,400 columns per node correspond to using about 11 and 177 compute nodes for a 1° and 0.25° CAM simulation, respectively). In addition, OpenACC performs better than OpenMP target offload, especially in a configuration where multiple MPI ranks run concurrently on the same GPU. A consistent behavior is observed in a practical CAM simulation that GPU is able to outperform CPU even at a coarse resolution. The speedup is up to 2.6× for a single NVIDIA V100 GPU and up to 3.6× for eight NVIDIA V100 GPUs, compared to the CPU performance of using 36 Intel Skylake CPU cores in a one-node simulation. An in-depth analysis of the GPU performance reveals that the data transfer between CPU and GPU dominates the overall GPU performance, which should be the focus to pursue a further speedup in the future. The speedup increases up to 5.4× in a CAM simulation at the high resolution (24 NVIDIA V100 GPUs against 108 Intel Skylake CPU cores), which highlights that we could take better advantage of GPU computing with larger problem size. Moreover, we demonstrate that using an algorithm with better parallelism (i.e., implicit sedimentation rather than explicit sedimentation) can markedly improve GPU performance, emphasizing the importance of choosing an appropriate algorithm for GPU computing. Last, we perform a series of CAM simulations with one MPI rank per GPU and it leads to a noticeably improved scalability of the GPU performance, compared to oversubscribing a GPU with multiple MPI ranks. Therefore, using one MPI rank per GPU is found to be an optimal configuration for GPU performance both in the standard-alone PUMAS kernel and practical CAM simulations.

This study has shown that it is possible to achieve a modest speedup by porting a single physics component in CAM from CPU to GPU while keeping the remainder of the code on the CPU. This demonstrates the viability of gradually porting additional physics components to GPU in the future. Enabling the execution of additional CAM physics packages on the GPU could not only save the computational time of these packages, but also the unnecessary data transfer since many components in CAM share the same data as input or output, which could result in large speedups. Once the whole CAM is ported to GPU, we will then be able to use the optimal configuration for GPU to perform a CAM simulation without compromising the performance on the CPU for those components that are not GPU-enabled yet. It will also be useful to evaluate the GPU performance of PUMAS (a) on the platforms from different vendors (e.g., NVIDIA A100 GPU or AMD MI250 GPU), (b) at a finer vertical resolution or (c) at an ultra-high horizontal resolution (e.g., 3.75 km targeted by the EarthWorks project), once those resources or configurations become available.

Data Availability Statement

The source code used in this study for CPU and GPU simulations, as well as the data for plotting, can be found at (Sun et al., 2022). The CAM source code is being actively developed at https://github.com/ESCOMP/CAM. The PUMAS source code is being actively developed at https://github.com/ESCOMP/PUMAS. The OpenMP target offload code for the PUMAS stand-alone kernel is generated by an auto-migration tool provided by the Intel Corporation (Servat et al., 2022).

Acknowledgments

This study is supported by the National Science Foundation (NSF) base funding of National Center for Atmospheric Research (NCAR). The authors acknowledge the funding support from NSF funded project Earthworks (NSF award number 531 2004973). This work is made possible from a number of contributions. In particular, the authors would like to thank the following: Jim Edwards from NCAR for his help on adding the GPU configuration to a CAM simulation on Casper, Steve Goldhaber and Cheryl Craig from NCAR for their help on the code development of CAM and PUMAS, and clarifications of some terminologies used in this study, Richard Loft from AreandDee LLC for his comments on the analysis of results, Raghu Raj Prasanna Kumar and Pranay Reddy Kommera from NVIDIA for their help on understanding the Nsight Systems profiling output and the general information about NVIDIA's GPU, Youngsung Kim from Oak Ridge National Laboratory (ORNL) for his detailed instructions about how to use KGEN to generate the PUMAS stand-alone kernel.

References

Abdi, D. S., Giraldo, F. X., Constantinescu, E. M., Carr, L. E., Wilcox, L. C., & Warburton, T. C. (2019). Acceleration of the implicit–explicit nonhydrostatic unified model of the atmosphere on manycore processors. *The International Journal of High Performance Computing Applications*, *33*(2), 242–267. https://doi.org/10.1177/1094342017732395

Bacmeister, J. T., Wehner, M. F., Neale, R. B., Gettelman, A., Hannay, C., Lauritzen, P. H., et al. (2014). Exploratory high-resolution climate simulations using the community atmosphere model (CAM). *Journal of Climate*, *27*(9), 3073–3099. https://doi.org/10.1175/JCLI-D-13-00387.1

Baker, A. H., Hammerling, D. M., Levy, M. N., Xu, H., Dennis, J. M., Eaton, B. E., et al. (2015). A new ensemble-based consistency test for the community Earth system model (pyCECT v1.0). *Geoscientific Model Development*, *8*(9), 2829–2840. https://doi.org/10.5194/gmd-8-2829-2015

Bertagna, L., Deakin, M., Guba, O., Sunderland, D., Bradley, A. M., Tezaur, I. K., et al. (2019). HOMMEXX 1.0: A performance-portable atmospheric dynamical core for the energy exascale Earth system model. *Geoscientific Model Development*, *12*(4), 1423–1441. https://doi.org/10.5194/gmd-12-1423-2019

Computational and Information Systems Laboratory. (2019). *Cheyenne: HPE/SGI ICE XA system (NCAR community computing)*. National Center for Atmospheric Research. https://doi.org/10.5065/D6RX99HX

Danabasoglu, G., Lamarque, J.-F., Bacmeister, J., Bailey, D. A., DuVivier, A. K., Edwards, J., et al. (2020). The community Earth system model version 2 (CESM2). *Journal of Advances in Modeling Earth Systems*, *12*(2), e2019MS001916. https://doi.org/10.1029/2019MS001916

Dennis, J. M., Edwards, J., Evans, K. J., Guba, O., Lauritzen, P. H., Mirin, A. A., et al. (2012). Cam-se: A scalable spectral element dynamical core for the community atmosphere model. *The International Journal of High Performance Computing Applications*, *26*(1), 74–89. https://doi.org/10.1177/1094342011428142

Fu, H., Liao, J., Xue, W., Wang, L., Chen, D., Gu, L., et al. (2016). Refactoring and optimizing the community atmosphere model (CAM) on the sunway taihulight supercomputer. In *Sc '16: Proceedings of the international conference for high performance computing, networking, storage and analysis* (pp. 969–980). https://doi.org/10.1109/SC.2016.82

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

Gettelman, A., Bardeen, C. G., McCluskey, C. S., Järvinen, E., Stith, J., Bretherton, C., et al. (2020). Simulating observations of southern ocean clouds and implications for climate. *Journal of Geophysical Research: Atmospheres*, *125*(21), e2020JD032619. https://doi.org/10.1029/2020JD032619

Gettelman, A., & Morrison, H. (2015). Advanced two-moment bulk microphysics for global models. Part I: Off-line tests and comparison with other schemes. *Journal of Climate*, *28*(3), 1268–1287. https://doi.org/10.1175/JCLI-D-14-00102.1

Gettelman, A., Morrison, H., Eidhammer, T., Thayer-Calder, K., Sun, J., Forbes, R., et al. (2023). Importance of ice nucleation and precipitation on climate with the parameterization of unified microphysics across scales version 1 (PUMASv1). *Geoscientific Model Development*, *16*(6), 1735–1754. https://doi.org/10.5194/gmd-16-1735-2023

Gettelman, A., Morrison, H., & Ghan, S. J. (2008). A new two-moment bulk stratiform cloud microphysics scheme in the community atmosphere model, version 3 (CAM3). Part II: Single-column and global results. *Journal of Climate*, *21*(15), 3660–3679. https://doi.org/10.1175/2008JCLI2116.1

Gettelman, A., Morrison, H., Santos, S., Bogenschutz, P., & Caldwell, P. M. (2015). Advanced two-moment bulk microphysics for global models. Part II: Global model solutions and aerosol–cloud interactions. *Journal of Climate*, *28*(3), 1288–1307. https://doi.org/10.1175/JCLI-D-14-00103.1

Gettelman, A., Morrison, H., Thayer-Calder, K., & Zarzycki, C. M. (2019). The impact of rimed ice hydrometeors on global and regional climate. *Journal of Advances in Modeling Earth Systems*, *11*(6), 1543–1562. https://doi.org/10.1029/2018MS001488

Guo, H., Ming, Y., Fan, S., Zhou, L., Harris, L., & Zhao, M. (2021). Two-moment bulk cloud microphysics with prognostic precipitation in GFDL's atmosphere model AM4.0: Configuration and performance. *Journal of Advances in Modeling Earth Systems*, *13*(6), e2020MS002453. https://doi.org/10.1029/2020MS002453

Harris, L., Zhou, L., Lin, S.-J., Chen, J.-H., Chen, X., Gao, K., et al. (2020). GFDL SHiELD: A unified system for weather-to-seasonal prediction. *Journal of Advances in Modeling Earth Systems*, *12*(10), e2020MS002223. https://doi.org/10.1029/2020MS002223

Kim, J. Y., Kang, J.-S., & Joh, M. (2021). GPU acceleration of MPAS microphysics WSM6 using OpenACC directives: Performance and verification. *Computers & Geosciences*, *146*, 104627. https://doi.org/10.1016/j.cageo.2020.104627

Kim, Y., Dennis, J., Kerr, C., Kumar, R. R. P., Simha, A., Baker, A., & Mickelson, S. (2016). KGEN: A python tool for automated Fortran Kernel generation and verification. *Procedia Computer Science*, *80*, 1450–1460. https://doi.org/10.1016/j.procs.2016.05.466

Mielikainen, J., Huang, B., Huang, H.-L. A., Goldberg, M. D., & Mehta, A. (2013). Speeding up the computation of WRF double-moment 6-class microphysics scheme with GPU. *Journal of Atmospheric and Oceanic Technology*, *30*(12), 2896–2906. https://doi.org/10.1175/JTECH-D-12-00218.1

Mielikainen, J., Price, E., Huang, B., Huang, H.-L. A., & Lee, T. (2016). GPU compute unified device architecture (CUDA)-based parallelization of the RRTMG shortwave rapid radiative transfer model. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, *9*(2), 921–931. https://doi.org/10.1109/JSTARS.2015.2427652

Milroy, D. J., Baker, A. H., Hammerling, D. M., Dennis, J. M., Mickelson, S. A., & Jessup, E. R. (2016). Towards characterizing the variability of statistically consistent community Earth system model simulations. *Procedia computer science* (Vol. 80, pp. 1589–1600). (International Conference on Computational Science 2016, ICCS 2016, 6-8 June 2016). https://doi.org/10.1016/j.procs.2016.05.489

Milroy, D. J., Baker, A. H., Hammerling, D. M., & Jessup, E. R. (2018). Nine time steps: Ultra-fast statistical consistency testing of the community Earth system model (pyCECT v3.0). *Geoscientific Model Development*, *11*(2), 697–711. https://doi.org/10.5194/gmd-11-697-2018

Morrison, H., & Gettelman, A. (2008). A new two-moment bulk stratiform cloud microphysics scheme in the community atmosphere model, version 3 (CAM3). Part I: Description and numerical tests. *Journal of Climate*, *21*(15), 3642–3659. https://doi.org/10.1175/2008JCLI2105.1

Morrison, H., Thompson, G., & Tatarskii, V. (2009). Impact of cloud microphysics on the development of trailing stratiform precipitation in a simulated squall line: Comparison of one- and two-moment schemes. *Monthly Weather Review*, *137*(3), 991–1007. https://doi.org/10.1175/2008MWR2556.1

Morrison, H., van Lier-Walqui, M., Fridlind, A. M., Grabowski, W. W., Harrington, J. Y., Hoose, C., et al. (2020). Confronting the challenge of modeling cloud and precipitation microphysics. *Journal of Advances in Modeling Earth Systems*, *12*(8), e2019MS001689. https://doi.org/10.1029/2019MS001689

Norman, M., Larkin, J., Vose, A., & Evans, K. (2015). A case study of CUDA FORTRAN and OpenACC for an atmospheric climate kernel. *Journal of Computational Science*, *9*, 1–6. https://doi.org/10.1016/j.jocs.2015.04.022

Servat, H., Rossi, G., Duran, A., & Narayanaswamy, R. (2022). On the migration of OpenACC-based applications into OpenMP 5+. In M. Klemm, B. R. de Supinski, J. Klinkenberg, & B. Neth (Eds.), *OpenMP in a modern world: From multi-device support to meta programming* (pp. 127–141). Springer International Publishing. https://doi.org/10.1007/978-3-031-15922-0_9

Sun, J., Dennis, J., Mickelson, S., Vanderwende, B., Gettelman, A., & Thayer-Calder, K. (2022). Accelerate the parameterization of unified microphysics across scales (pumas) on the graphics processing unit (GPU) with directive-based methods [Dataset]. https://doi.org/10.5281/zenodo.7324460

Sun, J., Fu, J. S., Drake, J. B., Zhu, Q., Haidar, A., Gates, M., et al. (2018). Computational benefit of GPU optimization for the atmospheric chemistry modeling. *Journal of Advances in Modeling Earth Systems*, *10*(8), 1952–1969. https://doi.org/10.1029/2018MS001276

Sun, J., Zhang, K., Wan, H., Ma, P.-L., Tang, Q., & Zhang, S. (2019). Impact of nudging strategy on the climate representativeness and hindcast skill of constrained EAMv1 simulations. *Journal of Advances in Modeling Earth Systems*, *11*(12), 3911–3933. https://doi.org/10.1029/2019MS001831

Taylor, M. A., & Fournier, A. (2010). A compatible and conservative spectral element method on unstructured grids. *Journal of Computational Physics*, *229*(17), 5879–5895. https://doi.org/10.1016/j.jcp.2010.04.008

Wang, Y., Zhao, Y., Li, W., Jiang, J., Ji, X., & Zomaya, A. Y. (2019). Using a GPU to accelerate a longwave radiative transfer model with efficient CUDA-based methods. *Applied Sciences*, *9*(19), 4039. https://doi.org/10.3390/app9194039

Zhang, S., Fu, H., Wu, L., Li, Y., Wang, H., Zeng, Y., et al. (2020). Optimizing high-resolution community Earth system model on a heterogeneous many-core supercomputing platform. *Geoscientific Model Development*, *13*(10), 4809–4829. https://doi.org/10.5194/gmd-13-4809-2020

19422466, 2023, 5, Downloaded from https://agupubs.onlinelibrary.wiley.com/doi/10.1029/2022MS003515 by Battelle Memorial Institute, Wiley Online Library on [21/11/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

Zheng, X., Klein, S. A., Ma, H.-Y., Bogenschutz, P., Gettelman, A., & Larson, V. E. (2016). Assessment of marine boundary layer cloud simulations in the cam with CLUBB and updated microphysics scheme based on arm observations from the Azores. *Journal of Geophysical Research: Atmospheres*, *121*(14), 8472–8492. https://doi.org/10.1002/2016JD025274
Zhou, L., Lin, S.-J., Chen, J.-H., Harris, L. M., Chen, X., & Rees, S. L. (2019). Toward convective-scale prediction within the next generation global prediction system. *Bulletin of the American Meteorological Society*, *100*(7), 1225–1243. https://doi.org/10.1175/BAMS-D-17-0246.1

Erratum

In the originally published version of this article, the first code in Section 2.4 contained an error. "dok = 1, nlev" and "doi = 1, ncol" should be "do k = 1, nlev" and "do i = 1, ncol." The errors have been corrected, and this may be considered the authoritative version of record.

19422466, 2023, 5, Downloaded from https://agupubs.onlinelibrary.wiley.com/doi/10.1029/2022MS003515 by Battelle Memorial Institute, Wiley Online Library on [21/11/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License