This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

SAND2020-8904C

# A Performance-Portable Nonhydrostatic Atmospheric Dycore for the Energy Exascale Earth System Model Running at Cloud-Resolving Resolutions.

Luca Bertagna
Sandia National Laboratories
PO BOX 5800, 87122
Albuquerque, NM, USA
lbertag@sandia.gov

Oksana Guba
Sandia National Laboratories
PO BOX 5800, 87122
Albuquerque, NM, USA
onguba@sandia.gov

Mark A. Taylor
Sandia National Laboratories
PO BOX 5800, 87122
Albuquerque, NM, USA
mataylo@sandia.gov

James G. Foucar
Sandia National Laboratories
PO BOX 5800, 87122
Albuquerque, NM, USA
jgfouca@sandia.gov

Jeff Larkin
NVIDIA Corporation
Santa Clara, CA, USA
jlarkin@nvidia.com

Andrew M. Bradley
Sandia National Laboratories
PO BOX 5800, 87122
Albuquerque, NM, USA
ambradl@sandia.gov

Sivasankaran Rajamanickam
Sandia National Laboratories
PO BOX 5800, 87122
Albuquerque, NM, USA
srajama@sandia.gov

Andrew G. Salinger
Sandia National Laboratories
PO BOX 5800, 87122
Albuquerque, NM, USA
agsalin@sandia.gov

## ABSTRACT

We present an effort to port the nonhydrostatic atmosphere dynamical core of the Energy Exascale Earth System Model (E3SM) to efficiently run on a variety of architectures, including conventional CPU, many-core CPU, and GPU. We specifically target cloud-resolving resolutions of 3 km and 1 km. To express on-node parallelism we use the C++ library Kokkos, which allows us to achieve a performance portable code in a largely architecture-independent way. Our C++ implementation is at least as fast as the original Fortran implementation on IBM Power9 and Intel Knights Landing processors, proving that the code refactor did not compromise the efficiency on CPU architectures. On the other hand, when using the GPUs, our implementation is able to achieve 0.97 Simulated Years Per Day, running on the full Summit supercomputer. To the best of our knowledge, this is the most achieved to date by any global atmosphere dynamical core running at such resolutions.

***Index Terms*—Atmospheric modeling, Multicore processing, High performance computing**

## I. JUSTIFICATION FOR GORDON-BELL PRIZE

We present an efficient and performance portable implementation of the Energy Exascale Earth System Model's nonhydrostatic atmosphere dynamical core. Using Summit's 27600 GPUs, we obtain record setting performance of 0.97 simulated years per day on a realistic cloud-resolving community benchmark.

## II. PERFORMANCE ATTRIBUTES

| Attribute title | Attribute value |
|---|---|
| Category achievement | Time-to-solution |
| Type of method used | Finite element with implicit/explicit time-stepping |
| Results reported on the basis of | Whole application excluding I/O and initialization |
| Precision reported | Double precision |
| System scale | Results measured on full-system scale |
| Measurement mechanism | Timers |

## III. PROBLEM OVERVIEW

Climate models are a critical tool used to evaluate the risks and impacts of climate change on society. The impacts are potentially devastating, but uncertainty in the predictions makes it difficult to assess the cost of inaction, mitigation, or adaptation. Models uniformly agree on the anthropogenic causes of global warming, but disagree on important details such as how much warming, how quickly it will occur, and changes to the frequency of extreme events [1]. Reducing the uncertainty in global climate simulations is one of the grand challenges for the climate modeling community [2], [3].

One of the major sources of uncertainty is due to the parameterized effects of convective cloud systems [4], [5]. Current climate models are run at horizontal resolutions (average grid spacing at the Equator) as fine as 25 km, which require the effects of these cloud systems to be approximated by *parameterizations*. Resolving these cloud systems starts to become possible at *cloud-resolving* resolutions in the range of 1-3 km [6], [7], the resolutions targeted in this paper.

The NICAM model was the first global cloud-resolving model (GCRM) capable of running global atmosphere sim-

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

ulations at 3.5 km resolution [8]. Since then, several modeling groups have developed GCRMs, such as the 9 models participating in the DYnamics of the Atmospheric general circulation Modeled On Nonhydrostatic Domains (DYAMOND) intercomparison project [9]. These models all require top-500 class CPU systems and obtain throughput rates in the range of several simulated *days* per wall clock day (SDPD). This throughput is suitable for short weather forecast type simulations, such as the 40 day simulation used in DYAMOND. But running GCRMs for century-long simulations will require throughput in the simulated *years* per wall clock day (SYPD) range. This throughput remains out of reach, but we are rapidly approaching this capability with a combination of new algorithms and upcoming exascale computing architectures.

Here we present a step towards this goal, and describe the performance of the atmosphere dynamical core (henceforth *dycore*) used in the GCRM being developed for the Energy Exascale Earth System Model (E3SM) project [10], [11]. The dycore solves the fully compressible Navier-Stokes equations, with molecular diffusion replaced by a turbulence model, and additionally transports several constituents, including multiple forms of water and various aerosols. It is typically the most expensive component of a GCRM, representing more than half the cost. It also contains all of a GCRM's inter-node communication, and is thus the most challenging component to run at scale on parallel computers.

We note that in a full Earth system model, there will be many additional concerns, one of the biggest being memory. For the E3SM, we often run each component model on independent nodes so that they each have access to the full memory of each node. For the atmosphere, this will entail running a suite of physical parameterizations in addition to the dynamical core described in this work. We have a Fortran version of this physics running on the KNL architecture, where we have run the full cloud resolving atmosphere model at 3 km resolution on 1536 nodes, each with 96 GB. We thus anticipate that atmosphere model will fit entirely in the GPU memory on Summit, where each Summit node contains 96 GB of GPU memory spread across 6 GPUs.

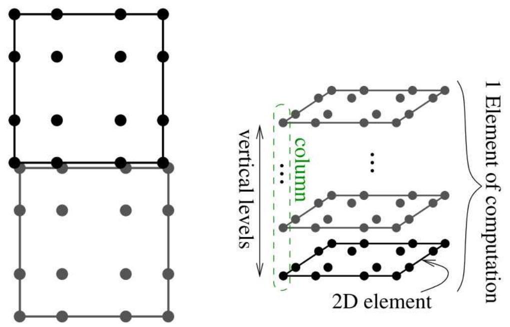
We present results for two cloud-resolving resolutions. The first contains 7.2 billion grid points, with a horizontal resolution of 3 km, where we use a standard community benchmark, allowing us to compare our performance with several other models, including a previous Gordon Bell finalist. The second has 65 billion grid points, with a 1 km horizontal resolution. With 16 unknowns per grid point, this finer resolution corresponds to over 1 trillion total degrees of freedom. For both problems, we use 128 vertical levels, equally spaced in pressure. This results in an average vertical resolution of 0.4 km, dropping to 0.05 km at the Earth's surface.

### A. The HOMMEXX-NH dycore

Our work is implemented in the High Order Methods Modeling Environment (HOMME) [12]–[14]. HOMME contains the Fortran-based hydrostatic dycore used by E3SM [15], [16] and the Community Earth System Model (CESM) [17], [18]. This dycore was recently rewritten in C++, using the Kokkos library [19] for on-node parallelism, in a work (henceforth, HOMMEXX) that showcased the ability of Kokkos to provide competitive or improved on-node performance on a variety of architectures including KNL and GPUs [20]. For cloud-resolving resolutions, the hydrostatic approximation used in HOMMEXX is not appropriate and we require a *nonhydrostatic* dynamical core [7]. Building on the success of [20], we have thus developed HOMMEXX-NH, a new implementation combining the performance portable approach of HOMMEXX and the nonhydrostatic formulation of the equations recently developed in [21].

We follow a common approach to nonhydrostatic modeling of solving the fully compressible Navier-Stokes equations written in a terrain following mass based vertical coordinate [22], [23]. This results in a set of prognostic equations for the three components of the velocity field, the mass-coordinate pseudo-density, the geopotential height and a thermodynamic variable, for which we use virtual potential temperature. We also transport an arbitrary number of additional species, representing water vapor, liquid and ice water, and various aerosols. The number of such tracers varies depending on the types of atmosphere parameterizations used. In the benchmark problems presented here, we always transport 10 species.

The prognostic equations consist of the time-reversible adiabatic terms from [21], combined with hyperviscosity following [13], [24]. For the adiabatic terms, we use a structure preserving formulation in order to preserve the discrete Hamiltonian and produce an energetically consistent model. The horizontal discretization uses the collocated mimetic spectral element method from [25], with conservative and monotone tracer transport [26]. The vertical discretization uses a Lorenz staggered extension of the mimetic centered difference from [27]. Figure 1 depicts the space discretization of HOMME. With a Lorenz staggering, prognostic variables are located at level midpoints, with the exception of the vertical velocity and geopotential, which are located at level interfaces. For the vertical transport terms, we continue to use HOMME's vertically Lagrangian approach adapted from [28].



(a) Two 2D neighbor elements

(b) Structure of one 3D element

Fig. 1. Horizontal and vertical structure of DOFs, marked as dots, in HOMME. As shown in (a), DOFs along the edges of 2D elements are duplicated on each element. In (b), 3D element consists of a stack of 128 2D elements, each with 16 DOFs (GLL) points.

For the temporal discretization, we use a Horizontally Explicit Vertically Implicit (HEVI) approach [29], discretized with an IMplicit-EXplicit (IMEX) Runge Kutta method [30]. The HEVI splitting decomposes the equations into a set of terms which represent vertically propagating acoustic waves, and the remaining terms which include all horizontal derivatives. While the latter can be treated explicitly, we treat the vertical acoustic waves implicitly, due to the significantly finer grid spacing in the vertical direction. This allows the model to use timesteps substantially larger than the ones required by a fully explicit method. We use a highly efficient IMEX method from [31], with a 2nd order accurate coupling of a high-stage high-CFL scheme for the explicit terms and a Diagonally Implicit Runge Kutta (DIRK) scheme for the implicit terms. Due to the use of the Laprise mass coordinate, the vertical acoustic waves are isolated to only two terms in the two equations solved at level interfaces, leading to an implicit system for a single variable. The details of the nonlinear solve are presented in Section V-A.

## IV. State of the art

### A. Overview of GCRMs

GCRMs have long relied on the world's fastest computers, with NICAM first running at near cloud-resolving resolutions on the Earth Simulator (Ranked #1 in 2002, [32]), followed by the first sub 1 km GCRM simulations [33] running on the K-computer (Ranked #1 in 2011). More recent results include Gordon Bell awardee [34] and finalist [35] running on the Sunway TaihuLight (Ranked #1 in 2016). Our work continues this trend, presenting results on Summit (Ranked #1 in 2019) and representing one of the first GCRMs to make use of GPU acceleration.

It is difficult to compare the performance of different GCRMs across different hardware since models can be configured with different horizontal and vertical resolutions, include different physical parameterizations and number of prognostic tracers. Similar difficulties were confronted during the U.S. National Weather Service's (NWS) process to select the atmosphere dynamical core for their next generation operational weather forecast model. To address these issues, the NWS established the Next Generation Global Prediction System (NGGPS) which evaluated several GCRMs [36], developed a standard dycore 3 km GCRM benchmark problem, and compared the performance of several GCRMs on the Edison supercomputer (Ranked #18 in 2014) [37]. Here we rely heavily on this NGGPS 3 km benchmark, and also present results from this benchmark at 1 km resolutions.

Despite Edison's age and lower ranking in the Linpack based top500 list, its Intel Xeon Ivy Bridge CPU performed quite well on the finite volume and finite element codes typical in GCRMs. The GCRM performance on Edison remains competitive, and exceeding this performance on today's pre-exascale architectures requires significant innovations. In the original NGGPS Edison benchmarks, the fastest result was obtained by the NMM-UJ dycore [38] at 0.34 SYPD in single precision. The FV3 dycore [28], obtained a throughput of 0.16 SYPD (double precision) and 0.26 SYPD (single precision). Weather forecast models are often run in single precision, which can improve throughput by $\approx 60\%$ compared to double precision. Here, we present only double precision results, as most climate models use double precision due to the length of the simulations and the importance of conservation properties.

| GCRM Model | Computer (LINPACK Benchmark) | NGGPS Benchmark |
|---|---|---|
| FV3 | Edison (2.6 PF) | 0.16 SYPD |
| HOMME | TaihuLight (125 PF) | 0.34 SYPD |
| HOMMEXX-NH | Summit (200 PF) | 0.97 SYPD |

TABLE I
NGGPS Standard 3 km GRCM dycore benchmark (double precision)

FV3's record level of performance was broken in a 2018 Gordon Bell submission [35]. In that work, a substantial rewrite of the HOMME dynamical core was able to achieve 0.34 SYPD (double precision). This was using the HOMME hydrostatic model and the performance will be slightly lower when this model is upgraded to the nonhydrostatic equations necessary for a GCRM. For example, in the NGGPS benchmark on Summit for 4096 nodes, HOMMEXX-NH is 27% slower than the hydrostatic HOMMEXX.

For our work, described in detail below, HOMMEXX-NH runs at a record setting 0.97 SYPD in the NGGPS benchmark. We summarize these results in Table I. The results span 6 years of hardware, software, and algorithm developments, and illustrate the difficulty of improving the performance of climate models on modern architectures. The Sunway TaihuLight is 48 times faster than Edison in terms of Linpack FLOPS, but only achieves a $2.1\times$ speedup in the NGGPS benchmark. We have been able to improve this ratio, where the $76\times$ faster Summit (Linpack FLOPS) is $6\times$ faster on the NGGPS benchmark.

### B. Regional Cloud-Resolving Models

There have been several efforts to run *regional* cloud-resolving atmosphere models at global scales. Two recent results include the 2016 Gordon Bell winner [34] and the COSMO effort [39]. These models are difficult to compare directly with GCRMs, since they would need to be adapted from running on logically Cartesian domains to grids which cover the entire Earth's surface. This requires addressing the *pole problem*, which presents several challenges which impact either computational performance or parallel scalability [40]. However, to put these results in perspective with the NGGPS benchmark results, we note that on the Sunway TaihuLight, [34] reported obtaining 1 SYPD at 3 km and 0.07 SYPD at 0.5 km resolution. They used a benchmark problem with the same number of vertical levels as the NGGPS benchmark (128), but on a domain that only covers 32% of the Earth's surface, and the transport of one water variable instead of 10. COSMO was one of the first regional models to be ported to a GPU system, and in [39] it is benchmarked on Piz Daint (Ranked #3 in 2016). They present results using a 80°S to 80°N domain (covering 98.4 % of the Earth's surface), with fewer vertical levels (60) and fewer tracer species (7), but significantly more complex physical parameterizations. At 1.9 km this configuration obtained 0.23 SYPD, and at 0.93 km resolution obtained 0.043 SYPD on Piz Daint supercomputer (Ranked #3 in 2016).

### *C. The Kokkos programming model*

Kokkos [19] is an open source C++ library (with C++11 standard required) for efficient on-node parallelism using threads. It uses template metaprogramming to create abstractions that allow the user to write a single code base, with Kokkos mapping the code to the underlying thread model (possibly using architecture-specific code).

In particular, Kokkos offers constructs for the most common parallel patterns (for, reduce, and scan), abstracting key concepts such as *execution space* (the threading model used, such as Cuda or OpenMP), or *execution policy* (how the work is going to be distributed among threads). The latter is particularly important when trying to expose maximum parallelism (a key performance requirement on GPU architectures). In Kokkos, a `RangePolicy` represents the overall work by a linear range of integers, and the kernel is executed for each integer in the range, with different threads processing different work items. On the other hand, with a `TeamPolicy`, available threads are grouped into *teams*, and each team is assigned a different work item. A range policy is usually preferable in case of tightly nested loops, while a team policy is preferable when there are multiple inner loops, where teams of threads can cooperate to perform shared work.

Kokkos also implements a multidimensional array, called `View`, built on abstractions such as *data type* (the C++ type of the data stored), *memory space* (where the data is physically stored on the hardware), *layout* (how the multidimensional array is mapped into a linear array in memory), and *memory traits* (how the data is meant to be handled/accessed).

We point out that Kokkos is just one of several libraries available for performance portability. Other notable ones are RAJA [41], Charm++ [42], OCCA [43], HEMI [44], and HPX [45]. Other approaches for performance portability relying on the compiler pre-processor are also worth mentioning, such as OpenACC [46], OpenMP [47], and the Claw DSL [48].

## V. INNOVATIONS REALIZED

A performant and portable version of hydrostatic HOMME, HOMMEXX, was introduced in [20]. Here we built on its success to develop HOMMEXX-NH, a nonhydrostatic version of HOMMEXX. Spherical differential operators, packing/unpacking, and MPI halo exchange routines implemented in [20] were reused here with minimal changes. Given the different set of equations solved in the nonhydrostatic model, some functors from HOMMEXX had to be adjusted or added. In particular, we introduced functors for implicit solves in an IMEX Runge-Kutta timestepping method.

In this work, following [20], we use cubed-sphere grids, where the surface of the sphere is tiled by quadrilateral elements, partitioned among MPI ranks. The surface of the sphere is decomposed into 6 panels (the faces of an inscribed cube), and then in each face we use an array of two-dimensional $n_e \times n_e$ spectral elements, for a total of $6n_e^2$ elements. An example of a low resolution mesh with 10 elements per cube edge ($n_e = 10$) is shown in Fig. 2. Each element contains a tensor product of $4 \times 4$ Gauss-Lobatto-Legendre (GLL) nodes with 128 levels in the vertical/radial direction, as depicted in Fig. 1.

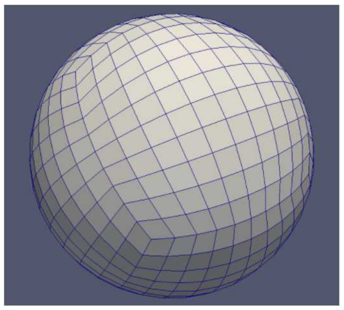


Fig. 2. Example of a cubed-sphere quadrilateral mesh with $n_e = 10$, yielding 600 elements.

When writing the new dycore, we used the same fundamental principles that drove the design choices in [20]. Namely, we highly leveraged Kokkos team policies to expose maximum parallelism, used packs of doubles to enhance vectorization on CPU, and carefully minimized memory movement. It is worth noticing that using Kokkos does not significantly increase the complexity of the GPU port effort, when compared with a raw CUDA approach. For the most part, the cost of using Kokkos is an increase in templates syntax (which can be partly hidden via usage of aliases and typedefs), and the consequent increase in compilation time. In our experience, the biggest challenge when porting an existing code to GPU is how to take advantage of the massive amount of parallelism offered by the device, which is not related to the approach used (Kokkos or raw CUDA), but rather to the structure and data dependency of the algorithm used. Nevertheless, in this context, Kokkos structures for hierarchical parallelism are helpful to clearly mark all the layers of parallelism while keeping the code syntax closer to the original algorithm, and are clearer than low-level checks on `threadIdx` and `blockIdx` structures in raw CUDA. Most critically for our project, we expect that the Kokkos abstraction will greatly reduce our effort in porting the code to subsequent architectures.

Compared with [20], refactoring the nonhydrostatic dycore presents a few additional challenges. The nonhydrostatic dycore requires an iterative solver of implicit equations that correspond to acoustic terms. On GPU, data dependencies in the solver make exposing maximum parallelism challenging. Efficient implementation of the solver on GPUs is crucial to achieve our performance results. Also, at very high resolutions, we need to manage GPU memory much more carefully than at low resolution. Finally, for the first time, the model is run at scale on a large GPU-based supercomputer. To efficiently run on Summit we searched for the most optimal set of parameters for parallel job execution and for MPI.

In the remainder of this section we highlight the handling of the nonlinear solver and the optimization of temporary buffers usage. These two components are the most innovative additions in this work and may be of interest to other projects.

## A. DIRK nonlinear solver

The vertically implicit part of the HEVI time integration method solves a nonlinear equation in three dimensions for vertical velocity $w$ and geopotential $\phi$ at interfaces. The equation must be solved at each stage of our DIRK time integration scheme. The equation involves derivatives only in the vertical direction; there is no coupling in the horizontal direction. Thus, the 3D equation decouples into many independent 1D equations, one for each vertical column. Let $n_{\text{lev}}$ be the number of vertical level midpoints in the model. Each 1D equation has $2n_{\text{lev}}$ unknowns for $w$ and $\phi$ but can be immediately rewritten to have just $n_{\text{lev}}$ unknowns. The Jacobian matrix of the resulting 1D nonlinear equation in $n_{\text{lev}}$ unknowns is tridiagonal and strictly diagonally dominant. Newton's method with a full Newton step is generally sufficient to solve the 1D equation. Occasionally, the step must be decreased to obtain a physically valid iterate, but this situation happens rarely and so the procedure is not described here. We use an analytical Jacobian that is factored at each Newton iteration. The solver termination criterion is a sufficiently small relative change in the iterate. In practice the average number of iterations required is 4, with the maximum never exceeding 10.

The original Fortran implementation uses LAPACK's general tridiagonal solvers DGTTRF and DGTTRS [49]. When developing our C++ implementation, we first proved that the Jacobian matrix is strictly diagonally dominant. We then exploited this fact in three ways. First, we use unpivoted solvers on both CPU and GPU. Second, an unpivoted solver can batch data across multiple 1D equations to obtain perfect vectorization on CPU architectures. Third, since pivoting is not required, on the GPU we can use cyclic reduction to expose substantial parallelism. We package our diagonally dominant tridiagonal solvers into a library. In the following, we discuss these points in further detail.

Because the data dependencies are purely in the vertical direction, it is optimal in both equation assembly and linear equation solution to arrange data so that the fastest index corresponds to GLL points in the horizontal direction. On a vector processor, GLL points are packed explicitly for vectorization. On the GPU, the lowest level of parallelism is along the GLL points. For example, since we use 16 GLL points per element, with AVX512 each horizontal 2D element packs into two 8-double packs, while on GPU it packs into half of a warp. This data layout requires transposes from the standard HOMMEXX layout into the DIRK solver layout and then out of it.

Because the Jacobian matrix is tridiagonal and strictly diagonally dominant, on the CPU, the Thomas algorithm, an unpivoted specialization of Gaussian elimination to tridiagonal systems, solves the equation efficiently. Our implementation packs along GLL points; thus, each arithmetic operation is a vector operation. For our problem, the Thomas algorithm can occupy up to 16 threads, one for each 1D equation in an element, which is not enough parallelism for the GPU. On the GPU, the solver must be able to use 16 warps—512 threads—effectively, since we use a thread block of 16 warps for each element. Cyclic reduction (CR) or the parallel cyclic reduction (PCR) variant are work inefficient but parallel efficient choices. With 512 threads and 16 deployed in each row, there is 32-way parallelism across rows. PCR is suboptimal relative to CR if it is not fully parallelized. Thus, we use CR. A hybrid approach, in which PCR is used to solve the 32-row CR subproblem [50], may be best, but our implementation uses pure CR.

The DIRK solver is divided into two kernels. A short kernel uses the standard HOMMEXX computational patterns to compute an initial guess. The second, main kernel runs the Newton iteration. Thus, repeated linear equation assembly and solution occur in one kernel. The following alternative would be necessary if we were to use, e.g., cuSPARSE's `cusparseDgtsv2StridedBatch` or `cusparseDgtsvInterleavedBatch`. In each Newton iteration, one kernel would assemble all equations for all elements. Then the solver would be called. Then a kernel would evaluate the step and termination criterion. The first and third could be fused. Thus, at minimum, there would be twice as many kernel invocations as Newton iterations, each sweeping across all elements. Our approach solves the equations in an element fully in two kernels total. On the CPU this approach has the obvious benefit of caching. On the GPU it naturally permits a flexible number of iterations per element, substantially fewer kernel invocations, and less persistent memory. The DIRK solver's code differs between GPU and CPU only in team thread and vector configuration and choice of tridiagonal linear equation solver from the solver library.

## B. Handling of temporary buffers

Following the design choices and patterns established in [20], the vast majority of the kernels of the nonhydrostatic dycore use a team policy. The kernels involved in the explicit and implicit steps of the DIRK method are particularly large, and require the calculation of several intermediate quantities. As a result, each team of threads ends up requiring a relatively large number of temporary buffers. A proper management of these buffers is crucial in order to keep the memory movement and usage limited.

The first solution we use to minimize memory usage is to use the same raw memory buffer for all the functors. In our application, the functor executions do not overlap, so that a functor is guaranteed to have completed its execution before the next one will start. An application using multiple CUDA streams would use one buffer per stream.

The second optimization is to make sure that each individual functor uses as little memory as possible. First, where possible, we rearranged the body of the functor so that each team of threads can use as few buffers as possible, by virtue of recycling the same buffer to store several intermediate quantities. Secondly, we implemented a handful of routines (which we collectively refer to as *workspace manager*, or WSM) that manage the use of buffers across thread teams, with the idea that, once a team has completed all its work on the current work item (usually, a single 3D element), its buffers can be *released*, and assigned to any other team, without the risk of overwriting important data. In other words, we only allocate enough buffers to accommodate the maximum number of teams that will be concurrently running ($\mathrm{NT}_{max}$). In particular, we need to be able to compute $\mathrm{NT}_{max}$ (or a reasonable upper bound), and to assign an id in $[0, \mathrm{NT}_{max})$ to each thread team, which we will then use to access buffers.

This task is relatively easy on CPU, where for each thread we can uniquely identify the team it belongs to, based on thread ID and team size. On GPU, it would be tempting to use the Cuda block ID as the team ID (recall that a team in Kokkos maps to a Cuda block). However, the block ID is bound by the number of blocks to be run, rather than the ones *concurrently* running. For large workload per node, this would yield a number larger than $\mathrm{NT}_{max}$.

Instead, we can get a good upper bound for $\mathrm{NT}_{max}$ as $\mathrm{NT}_{max} = (\#SM \times (threads/SM)_{max})/team\_size$. On a V100 GPU (which are the GPUs on the Summit supercomputer), we have 80 Streaming Multiprocessors (SM), with a maximum of 1024 threads per SM. A typical team size for our kernels is 512, which means we only need $(80 \times 1024)/512 = 160$ copies of each buffer to accommodate all teams. An array of integers is used to mark team IDs as currently in use or available. The WSM loops over this array (accessing it atomically) to find an available team id, and, upon work completion, it releases the team id. For performance reasons, we also add an *over-provision* factor, so that the number of buffer *slots* provided by the WSM is slightly larger, speeding up the search for an available team ID. By doing several trials, we concluded that a factor of 25% was a reasonable compromise between the need of increasing speed (obtained by increasing the over-provision factor) and the need to keep memory usage limited. In fact, increasing this factor above 25% did not appear to significantly affect how fast the WSM was able to retrieve an available team ID.

In Fig. 3 we show the single-node performance of some key functors, as well as the main time stepping loop, with and without the WSM enabled (at compile time), for a handful of different values of $n_e$. The metric reported is thousands of 3D elements processed per second. The version with WSM disabled (in blue) runs out of memory for $n_e > 21$, while the version with WSM enabled was able to accommodate also $n_e = 24$ and 30.

It is important to notice that the WSM comes at a runtime cost even at low $n_e$, where we would have enough memory to accommodate a separate buffer for each work item (rather than each team). Therefore, we turn this logic on at compile time only if we know our runtime configurations will yield a workload per node large enough to trigger this need.

It is worth mentioning that the memory reduction of WSM was crucial to be able to run the 1 km benchmark on as little as 2048 summit nodes (see Fig. 6).

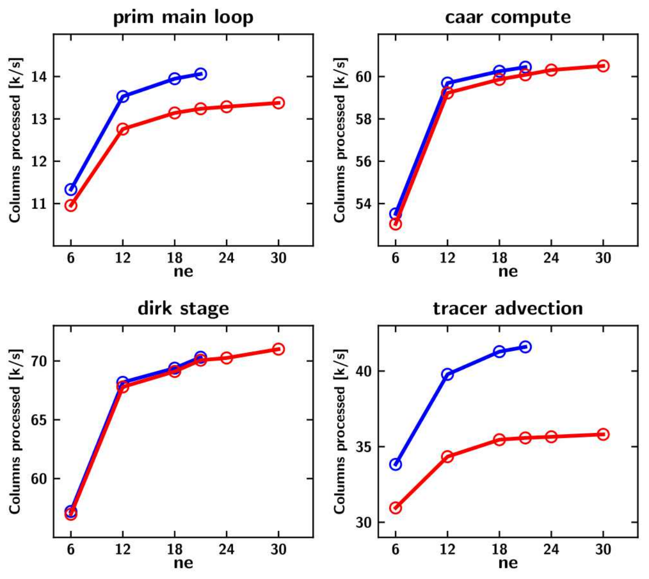


Fig. 3. Processing rate of key functors as well as main time loop timer for the workspace manager (WSM) enabled (red) and disabled (blue).

## VI. How performance was measured

### A. Full system performance

The throughput of the overall application is measured in the unit Simulated Years Per (wallclock) Day (SYPD), with 365 days in a simulated year. This is an intrinsic version of the time-to-solution metric, which permits evaluating the performance independently of the time horizon used in the simulation. Initialization time is excluded, and diagnostics and file I/O are turned off. The elapsed wallclock time is the maximum value across ranks of the timer over the top-level time-stepping loop. HOMME relies on the GPTL library [51] for wallclock time measurements.

### B. Individual kernels performance

For the purpose of this performance study the NVIDIA Tools Extensions (NVTX) API was added to the GPTL library to enhance GPU profiles with application-based annotations. Additionally the Kokkos Tools [52] were used to aid in kernel profiling. Additionally, the NVIDIA Nsight tools were used to gather performance metrics related to the GPU. NVIDIA Nsight Systems is a node-level tracing tool that gathers high-level information about the relative importance of various GPU kernels, NVLINK and PCIe data transfers, and CPU utilization. Once a high-level analysis is performed, NVIDIA Nsight Compute is used to gather the performance characteristics of individual kernels. Since gathering performance counters is an expensive operation, requiring kernels be rerun multiple times, only the first instance of each kernel was profiled and only for kernels that run for more than a few percent of the total GPU runtime.

Runtime percentages and kernel performance metrics were gathered on a problem configuration with mesh refinement level of $n_e = 16$, on a single node of Summit. This configuration results in the same workload per GPU of the full $n_e = 1024$ grid when run on 4096 nodes, but allows iterating on performance experiments more quickly without wasting computing time on full machine runs.

Kernel performance analysis in this paper will use the roofline model [53] [54], which presents the performance of individual kernels relative to the peak memory bandwidth and peak floating point operations on the GPU. The diagonal line on the plot represents memory bandwidth and the horizontal line represents peak floating point performance. The dots on the graph represent arithmetic intensity, the ratio of measured floating point operations to memory operations. The closer to one of these lines a dot appears, the more bounded by bandwidth or FLOPS a kernel is, with kernels never able to appear above either line, but may appear below the lines if other performance limiters apply, such as memory latency. Figure 8 shows the roofline analysis for the key kernels in the application, as measured by Nsight Compute.

### C. Architecture details

The results presented in this paper were obtained on the Summit supercomputer [55], [56], located at the Oak Ridge Leadership Computing Facility (OLCF) at Oak Ridge National Laboratory (ORNL), and on the NERSC Cori-KNL partition, located at the Lawrence Berkeley National Laboratory (LBNL).

Summit is based on the IBM AC 922 system architecture. It contains 4608 compute nodes, each constructed with 2 IBM Power 9 CPUs and 6 NVIDIA® Tesla® V100 GPUs with 512GB of DDR4 CPU memory and 16GB of HBM2 memory per-GPU. Summit has a Mellanox EDR 100G InfiniBand interconnect between nodes and an NVIDIA NVLINK® interconnect among GPUs, arranged in two groups of three GPUs. The Tesla V100 GPUs have a peak double precision floating point performance of 7.8 TFLOPS and a peak bandwidth of 900 GB/s.

Cori-KNL is based on the Intel Xeon Phi (Knights Landing, or KNL) 7250 processor. It contains 9688 compute nodes, each containing a single socket KNL processor, with 68 cores organized into 34 *tiles*, each comprising two cores sharing a 1MB L2 cache. Each core has 2 512-bit vector processing units, 4 hardware threads, and has a theoretical peak performance of 44.8 GFLOPS. Each node is equipped with 96GB of DDR4 2400 MHz memory, with 109GB/s peak bandwidth, as well as 16 GB of MCDRAM (multi-channel DRAM), with 460 GB/s. Cori-KNL has a Dragonfly topology Cray Aries interconnect.

### D. Verification of the refactored code

The dynamical core implemented in the original Fortran code was verified using idealized dynamical core tests, including the baroclinic instability test [57] and a suite of test cases from the Dynamical Core Model Intercomparison Project (DCMIP) [58], and convergence studies [21], [31]. Validation of the model implemented in the original Fortan code is out of the scope of this project. We focus on verification of the performance-portable code using the Fortran code as the base.

Testing is a crucial part of our refactoring process. In the early stages of development, we set up a harness of unit tests to verify individual refactored kernels using random inputs against their Fortran counterparts. Once the refactored code neared completion, we set up the baroclinic instability and DCMIP test cases to confirm end-to-end correctness of the new code.

We test the code in two regimes: bit-for-bit (BFB) and non-bit-for-bit (nonBFB). The former is used for testing and development, while the results presented in Section VII were obtained with the latter.

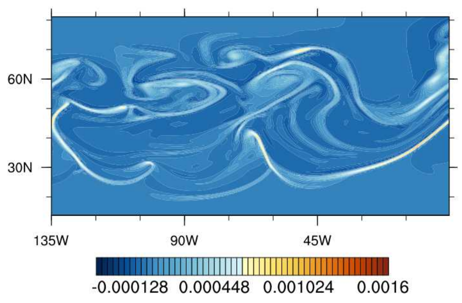
For the BFB regime we use the following optimization flags: `-fp-model strict -O0` for Intel compilers, `-ffp-contract=off -O2` for GNU compilers, and `-fmad=false` for nvcc (the NVIDIA CUDA compiler). In addition, in refactored GPU code, scans and reductions are serialized to allow BFB comparison with the original CPU code. For the nonBFB regime we use the following optimization flags: `-fp-model fast -O3 -DNDEBUG` for Intel, `-O3 -DNDEBUG` for GCC, and `-fmad=true` for nvcc.

To ensure correctness of the refactored code, in the BFB regime we run both unit tests and test cases. Additionally, since the two regimes contain different implementations for scans and reductions, we also verified the nonBFB regime by running the dry baroclinic instability test case at 26 km resolution for 15 days with timestep 70 sec (18514 total timesteps) on Summit, with the original Fortran code launched on CPUs and the refactored code launched on GPUs. Figure 4(a) contains a sample output of the relative vorticity field at day 15 for the refactored code. In 4(b), we plot the difference between the Fortran code and the refactored code. Overall, differences are of order 1e-11. In addition, for both runs we compared differences in surface pressure, pseudodensity, relative vorticity, nonhydrostatic pressure, potential temperature, velocity, and geopotential using `cprnc`, a common tool to evaluate Netcdf output for climate applications (shipped within the E3SM software). The biggest normalized Root Mean Square (RMS) error between the two codes is of order 1e-7. Considering the chaotic nature of the model, such small errors in the output after long time integration confirm correctness of the refactored code.

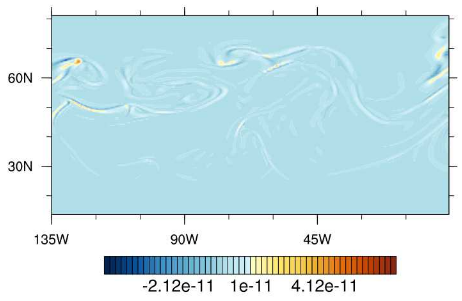
## VII. Performance Results

### A. Problem Configuration

We use the NGGPS benchmark [37]. This benchmark was originally designed to evaluate dycores for cloud resolving weather forecast models, and includes a 12 km and 3 km problem. The benchmark requires the dycore to be configured exactly how it would be run in a realistic weather forecast model or climate model. This includes the vertical resolution (128 levels) and timesteps of all the various subcomponents. The benchmark includes 10 tracers, which is fewer than what



(a) Refactored code



(b) Difference

Fig. 4. Baroclinic instability test case. (a) Relative vorticity computed by HOMMEXX-NH. (b) Difference of relative vorticity between the original and refactored codes.

is used by low resolution climate models, but is typical of GCRM models which typically do not include aerosol indirect effects. The tracers are initialized with a 0/1 checkerboard pattern to ensure that the tracer monotonicity limiters are active. The dynamics are initialized in a geostrophically balanced flow with a small perturbation that triggers a realistic baroclinic instability [59]. As an example, in Fig. 5 we show a snapshot of the water vapor after the baroclinic instability is fully developed.

For our 1 km benchmark, we keep all details of the 3 km benchmark and only change the horizontal resolution to 1 km. For HOMMEXX-NH, at 3 km, $n_e = 1024$ with a 10 s dynamics and tracer transport timestep and a 20 s remap timestep. At 1 km, $n_e = 3072$ with a 3.125 s dynamics and tracer transport timestep and a 6.25 s remap timestep. For both resolutions, there is one application of hyperviscosity per dynamics and tracer timestep.

## B. Scaling studies

Figure 6 shows our main result, strong scaling of the dycore in terms of the SYPD metric, for different implementations and for different resolutions. The grey line represents the slope of a perfect scaling. Numerical values for Fig. 6 are reported in Table II.

The MPI and threads configurations for runs in Fig. 6 are as follows. In CPU-only runs on Summit (marked with '3 km, CPU, Fortran' and '3 km, CPU, C++'), each node is assigned 42 MPI ranks, and each rank uses four OpenMP threads (SMT4). In GPU runs on Summit (marked with '3 km, GPU, C++' and '1 km, GPU, C++'), each node is assigned 6 MPI ranks, and each rank is associated with a GPU; OpenMP threads are not used. Finally, in KNL runs on Cori (marked

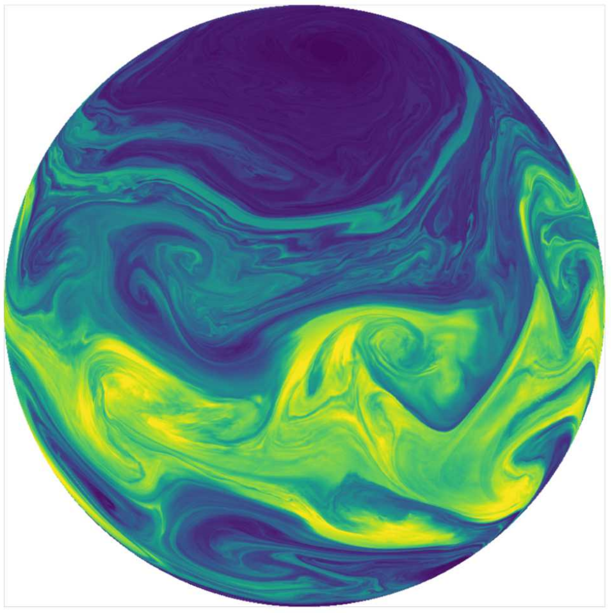


Fig. 5. Turbulent eddies in water vapor on the 500 hPa model layer from a HOMME-NH 3 km horizontal resolution simulation of baroclinic instability.

| Node count | Summit | | | | Cori | |
|---|---|---|---|---|---|---|
| | P9 | | GPU | | KNL | |
| | Fortran | C++ | C++ | C++ | C++ | Fortran |
| | 3 km | 3 km | 3 km | 1 km | 3 km | 3 km |
| 512 | 0.010 | 0.011 | 0.16 | | 0.012 | 0.008 |
| 1024 | 0.021 | 0.022 | 0.31 | | 0.25 | 0.018 |
| 1536 | 0.030 | 0.033 | 0.42 | | | |
| 2048 | 0.041 | 0.043 | 0.54 | 0.022 | 0.05 | 0.035 |
| 3072 | 0.055 | 0.064 | 0.71 | 0.033 | | 0.054 |
| 4096 | | | 0.90 | 0.044 | | 0.064 |
| 4600 | | | 0.97 | 0.049 | | |
| 8192 | | | | | | 0.139 |
| 9216 | | | | | 0.152 | 0.15 |

TABLE II
SYPD METRIC FOR DIFFERENT RUN CONFIGURATIONS. THE SAME DATA ARE PLOTTED IN FIG. 6.

with '3km, KNL, Fortran' and '3km, KNL, C++'), each node is assigned 64 MPI ranks, and each ranks uses 2 OpenMP threads.

HOMMEXX-NH was run on CPU and GPU for the 3 km and 1 km benchmarks. The original Fortran code was run on CPU and only for the 3 km benchmark. We did not run the 1 km benchmark on CPU since its performance in the SYPD metric would be too poor for practical applications. Note that the 1 km benchmark would not fit in fewer than 2048 Summit nodes, and it requires enabling the WSM for the 2048- and 3072-nodes runs.

The results show a perfect scaling on CPU of the portable C++ code, with a performance comparable or slightly better than the original Fortran implementation. On KNL, the improved performance of the C++ implementation over the Fortran one is more noticeable, because of the very efficient vectorization (a fact which is even more visible when focusing

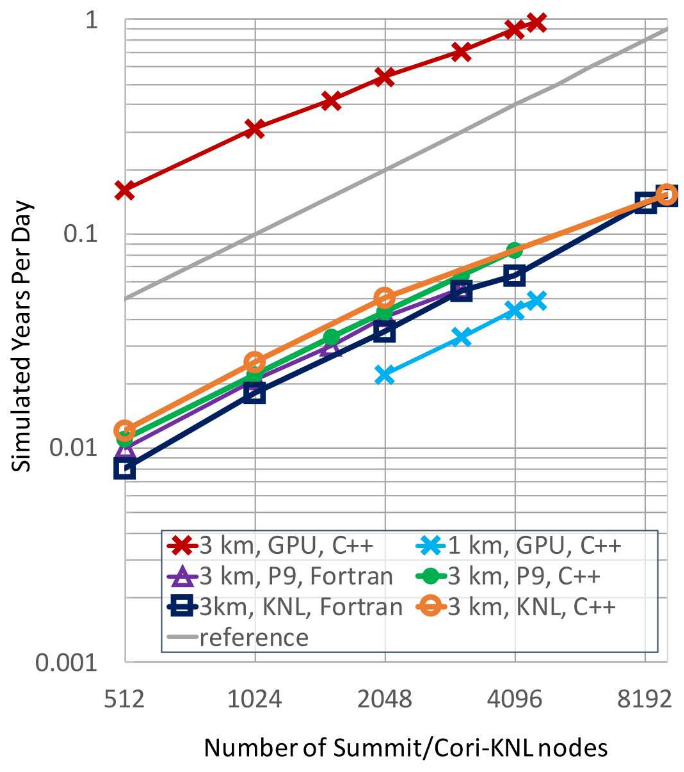


Fig. 6. Achieved SYPD for different implementations and resolutions. HOMMEXX-NH obtains a substantial acceleration when running on the GPU, while maintaining excellent performance when compared with the original Fortran implementation on the IBM P9 CPU and the Intel KNL CPU.

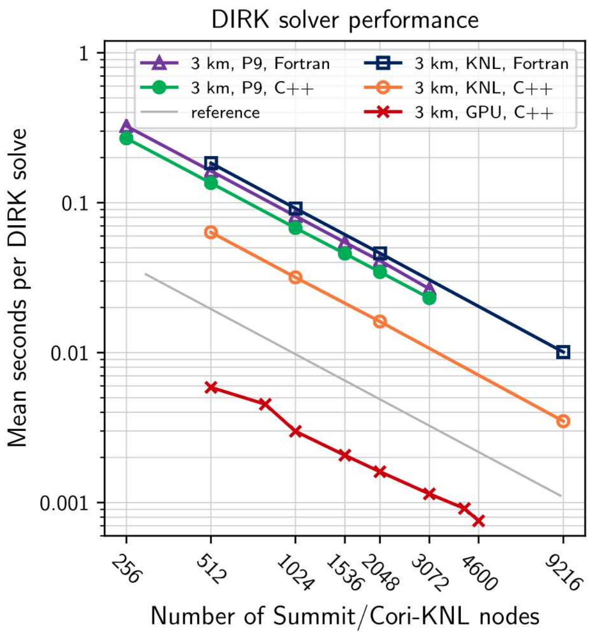


Fig. 7. Performance of the DIRK solver on Cori KNL, Summit Power9, and Summit V100 GPU, for the 3 km benchmark. The x-axis is the number of nodes used; in the case of GPUs, 6 GPUs are used per node. Performance, y-axis, is measured as mean time in seconds per DIRK solver call.

on DIRK performance; see below). The two implementations have similar performance when scaling up to the full Cori KNL partition (9216), likely because of the decreased workload per rank, but still maintain an excellent scaling. On GPU, our implementation scales perfectly at 1 km resolution. At 3 km resolution, at higher node counts scalability is reduced. The reason for this loss of performance is linked to the reduction of workload on each GPU. In a 512-nodes run, each GPU is assigned 2048 3D elements (i.e., a 2D element extruded vertically with 128 levels, see Fig. 1), while in a full system 4600-nodes run, each GPU is assigned at most 228 3D elements (the load balancing is not perfect with 4600 nodes), a reduction of about 88%. If we consider team (block, in CUDA language) sizes of 512 threads (as it is common in our implementation), this means each SM is processing at most three 3D elements. At this workload, the GPU run time becomes dominated by MPI communication costs. Nevertheless, even with a small workload per node, our implementation achieves 0.97 SYPD, which is roughly 67% of the value we could expect if the performance at 512 nodes was scaled linearly to 4600 nodes.

Figure 7 focuses on the performance results of the DIRK solver routines, for both the original (F90) and performance-portable (C++) implementations. First, on Cori KNL the C++ solver is at least 2.85 times faster than the F90 solver. This speedup is consistent with the tridiagonal solver's perfect vectorization; see Fig. 7 of [20] for more on the performance effect of perfect vectorization on KNL. Second, on Power9, the C++ solver is 1.15–1.2 times faster than the F90 solver. It is faster by only a little for two reasons: first, the C++ solver must perform data transposes that the F90 solver does not; second, Power9 has 128-bit rather than 512-bit vector instructions, reducing the maximum speedup of vectorized code. Third, comparing the two Power9 CPUs per Summit node against one (not six) V100 GPU, the C++ DIRK solver is 3.2–3.85 times faster on GPU than on CPU and 3.9–4.6 times faster than the F90 solver. Finally, on one V100 GPU, the solver is approximately 1.75 times faster than the C++ solver on a KNL node. In this final comparison, the GPU speedup is less than what we often expect with GPU because of the extremely effective vectorization on KNL.

## C. Individual kernel performance evaluation

For the defined benchmark problem, four Kokkos kernels occupy at least 5% of the application runtime and account for roughly 60% of the application runtime, so we will focus our analysis on these top kernels. Figure 8 shows the roofline analysis for the top three application kernels on the GPU, excluding recv_and_unpack because it has no arithmetic operations. We see that each of these kernels trends toward the memory bandwidth lines, indicating that their performance is limited more by available bandwidth than floating point performance.

We see in Table III that the TagPreExchange kernel, which is a portion of the explicit RK stages for dynamics, is roughly 20% of GPU runtime and has an arithmetic intensity of 0.69, indicating that it performs more memory operations than

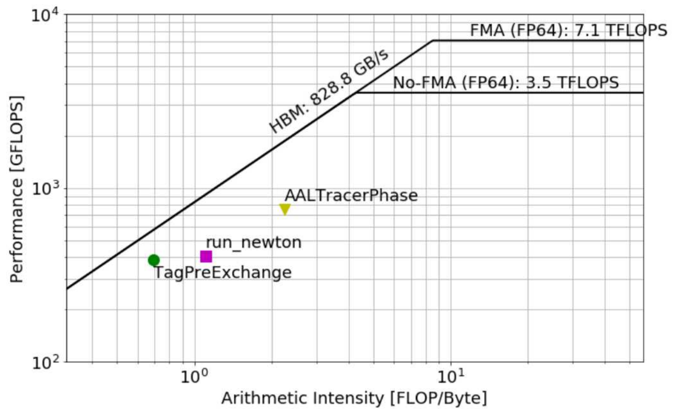


Fig. 8. Roofline Analysis for top kernels on a single V100 GPU.

mathematical operations. It achieves greater than 60% of peak memory bandwidth using no device-specific optimizations that would reduce portability to a CPU. The next most critical kernel is the run_newton kernel, the main DIRK solver kernel, which has an arithmetic intensity of 1.11, indicating more arithmetic operations than memory operation, but comes in slightly below the bandwidth roofline, indicating that it is likely bounded by memory latency. The occupancy of this kernel is just 25% due to the large amount of state carried throughout the lifetime of the kernel. Lower occupancy reduces the ability to hide memory latencies effectively, but breaking the function in hopes of increasing occupancy runs a risk of reduced cache efficiency on CPU architectures and may actually increase traffic to memory. The AALTracers kernel, a portion of the tracers advection timestep, is the next performance critical kernel, which has an arithmetic intensity of 2.24 FLOPs/byte and utilizes nearly 70% of the available GPU memory bandwidth. Next is the receive_and_unpack kernel, which handles unpacking MPI buffers during the boundary exchange. This operation inherently requires sub-optimal memory access patterns, which would require very device-specific optimizations to improve. Since no floating point operations are performed during this step, the arithmetic intensity is zero, but the kernel is able to utilize greater than 70% of available memory bandwidth despite its inherently poor memory access pattern along two sides of each element boundary. Overall, the majority of the significant application kernels are able to achieve a high degree of GPU memory utilization without giving up portability to device-specific optimizations in CUDA. Further performance improvements may be possible at a cost of the portability provided by the Kokkos framework.

| Kokkos Parallel Kernels by Importance | | | | |
|---|---|---|---|---|
| Function | CPU % | GPU % | GPU AI | GPU BW % |
| TagPreExchange | 21.9% | 23.6% | 0.69 | 63.5% |
| run_newton | 21.3% | 20.0% | 1.11 | 40.96% |
| AALTracerPhase | 11.2% | 12.8% | 2.24 | 68.02% |
| recv_and_unpack | 4.7% | 7.0% | 0.0 | 71.98% |
| Total Percentage | 59.1% | 63.4% | | |

TABLE III
PERFORMANCE OF TOP KOKKOS KERNELS.

## VIII. IMPLICATIONS

We presented a performance portable implementation of the nonhydrostatic atmosphere dynamical core of E3SM, and we evaluated the performance when running at cloud-resolving resolutions of 1 km and 3 km. Our implementation, which leverages Kokkos for on-node parallelism, is able to achieve 0.97 Simulated Years Per Day (SYPD), when running on GPU on the full Summit supercomputer. To the best of our knowledge, this is close to 3 times faster than what has been achieved by any other GCRM at such resolutions. Furthermore, by relying on Kokkos, our implementation is portable to different architectures, and will be able to rapidly capitalize on upcoming exascale machines while remaining competitive on traditional CPU platforms. The impact of this work could go beyond climate simulations. By demonstrating a portable application at a full system scale and at Gordon Bell level performance, this work lays the path forward for computational science applications to exploit several accelerator hardware types without rewriting the entire application for each one of them. Additionally, on a more practical level, during this work we developed two important features, the library for diagonally dominant tridiagonal linear systems and the workspace manager, which are sufficiently general to be used (directly or as a template) by other applications.

The success of this work is the foundation on which other parts of the E3SM next generation atmosphere model will build. In particular, we expect physics parameterizations to be implemented in this framework within a year. This will make it possible to run some of the first decade long cloud-resolving climate simulations. Such simulations are essential for scientists to fully address the problems arising from approximations in cloud systems used universally in traditional climate models. In the exascale era, these types of simulations could replace traditional climate models and remove one of the major sources of climate prediction uncertainty.

**Acknowledgements.** This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231.

Sandia National Laboratories is a multi-mission laboratory managed and operated by the National Technology and Engineering Solutions of Sandia, L.L.C., a wholly owned subsidiary of Honeywell International, Inc., for the DOE's National Nuclear Security Administration under contract DE-NA-0003525. SAND2020-0000.

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering, and early testbed platforms, in support of the nation's exascale computing imperative.

This research was supported as part of the Energy Exascale Earth System Model (E3SM) project, funded by the U.S. Department of Energy, Office of Science, Office of Biological and Environmental Research

This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

## REFERENCES

[1] R. K. Pachauri, L. Meyer, G.-K. Plattner, and T. Stocker, "Synthesis report. Contribution of working groups I, II and III to the fifth assessment report of the Intergovernmental Panel on Climate Change," *Intergovernmental Panel on Climate Change: Geneva, Switzerland*, 2014.

[2] T. Palmer, "Climate forecasting: Build high-resolution global climate models," *Nature News*, vol. 515, no. 7527, p. 338, 2014.

[3] T. Palmer and B. Stevens, "The scientific challenge of understanding and estimating climate change," *Proceedings of the National Academy of Sciences*, vol. 116, no. 49, pp. 24 390–24 395, 2019.

[4] B. Stevens and S. Bony, "What are climate models missing?" *Science*, vol. 340, no. 6136, pp. 1053–1054, 2013.

[5] S. C. Sherwood, S. Bony, and J.-L. Dufresne, "Spread in model climate sensitivity traced to atmospheric convective mixing," *Nature*, vol. 505, no. 7481, pp. 37–42, 2014.

[6] A. A. Wing, K. A. Reed, M. Satoh, B. Stevens, S. Bony, and T. Ohno, "Radiative-convective equilibrium model intercomparison project," *Geoscientific Model Development*, pp. 793–813, 2018.

[7] M. Satoh, B. Stevens, F. Judt, M. Khairoutdinov, S.-J. Lin, W. M. Putman, and P. Düben, "Global cloud-resolving models," *Current Climate Change Reports*, vol. 5, no. 3, pp. 172–184, May 2019.

[8] H. Tomita, H. Miura, S. Iga, T. Nasuno, and M. Satoh, "A global cloud-resolving simulation: Preliminary results from an aqua planet experiment," *Geophysical Research Letters*, vol. 32, no. 8, 2005.

[9] B. Stevens *et al.*, "DYAMOND: the DYnamics of the Atmospheric general circulation Modeled On Non-hydrostatic Domains," *Progress in Earth and Planetary Science*, vol. 6, no. 1, Sep. 2019.

[10] J.-C. Golaz *et al.*, "The DOE E3SM coupled model version 1: Overview and evaluation at standard resolution," *J. Adv. Model Earth Sy.*, vol. 11, no. 7, pp. 2089–2129, Mar. 2019.

[11] Energy Exascale Earth System Model. [Online]. Available: https://e3sm.org/

[12] J. Dennis, A. Fournier, W. F. Spotz, A. St.-Cyr, M. A. Taylor, S. J. Thomas, and H. Tufo, "High resolution mesh convergence properties and parallel efficiency of a spectral element atmospheric dynamical core," *Int. J. High Perf. Comput. Appl.*, vol. 19, pp. 225–235, 2005.

[13] J. Dennis, J. Edwards, K. Evans, O. Guba, P. Lauritzen, A. Mirin, A. St-Cyr, M. A. Taylor, and P. H. Worley, "CAM-SE: A scalable spectral element dynamical core for the community atmosphere model," *Int. J. High Perf. Comput. Appl.*, vol. 26, pp. 74–89, 2012.

[14] K. Evans, P. Lauritzen, S. Mishra, R. Neale, M. Taylor, and J. Tribbia, "AMIP simulation with the CAM4 spectral element dynamical core," *J. Climate*, vol. 26, no. 3, pp. 689–709, 2013.

[15] P. J. Rasch *et al.*, "An overview of the atmospheric component of the Energy Exascale Earth System Model," *J. Adv. Model Earth Sy.*, vol. 11, no. 8, pp. 2377–2411, 2019.

[16] P. M. Caldwell *et al.*, "The DOE E3SM coupled model version 1: Description and results at high resolution," *J. Adv. Model Earth Sy.*, vol. 11, no. 12, pp. 4095–4146, Dec. 2019.

[17] R. J. Small *et al.*, "A new synoptic scale resolving global climate simulation using the Community Earth System Model," *Journal of Advances in Modeling Earth Systems*, vol. 6, no. 4, pp. 1065–1094, Dec. 2014.

[18] S. Zhang *et al.*, "Optimizing high-resolution Community Earth System Model on a Heterogeneous Many-Core Supercomputing Platform (CESM-HR_SW1.0)," *Geoscientific Model Development Discussions*, vol. 2020, pp. 1–38, 2020. [Online]. Available: https://www.geosci-model-dev-discuss.net/gmd-2020-18/

[19] H. Carter Edwards, C. R. Trott, and D. Sunderland, "Kokkos: Enabling manycore performance portability through polymorphic memory access patterns," *J. Parallel Distr. Com.*, vol. 74, no. 12, pp. 3202–3216, 2014.

[20] L. Bertagna, M. Deakin, O. Guba, D. Sunderland, A. M. Bradley, I. K. Tezaur, M. A. Taylor, and A. G. Salinger, "HOMMEXX 1.0: a performance-portable atmospheric dynamical core for the Energy Exascale Earth System Model," *Geosci. Model Dev.*, vol. 12, no. 4, pp. 1423–1441, 2019.

[21] M. A. Taylor, O. Guba, A. Steyer, P. A. Ullrich, D. M. Hall, and C. Eldrid, "An energy consistent discretization of the nonhydrostatic equations in primitive variables," *J. Adv. Model Earth Sy.*, vol. 12, no. 1, 2020.

[22] A. Kasahara, "Various vertical coordinate systems used for numerical weather prediction," *Mon. Weather Rev.*, vol. 102, pp. 509–522, 1974.

[23] R. Laprise, "The Euler equations of motion with hydrostatic pressure as an independent variable," *Mon. Weather Rev.*, vol. 120, no. 1, pp. 197–207, 1992.

[24] O. Guba, M. Taylor, P. Ullrich, J. Overfelt, and M. Levy, "The spectral element method on variable resolution grids: Evaluating grid sensitivity and resolution-aware numerical viscosity," *Geosci. Model Dev.*, vol. 7, pp. 4081–4117, 2014.

[25] M. A. Taylor and A. Fournier, "A compatible and conservative spectral element method on unstructured grids," *J. Comput. Phys.*, vol. 229, pp. 5879– 5895, 2010.

[26] O. Guba, M. Taylor, and A. St.-Cyr, "Optimization based limiters for the spectral element method," *J. Comput. Phys.*, vol. 267, pp. 176–195, 2014.

[27] A. J. Simmons and D. M. Burridge, "An energy and angular momentum conserving vertical finite-difference scheme and hybrid vertical coordinates," *Mon. Weather Rev.*, vol. 109, pp. 758–766, 1981.

[28] S.-J. Lin, "A vertically Lagrangian finite-volume dynamical core for global models," *Mon. Weather Rev.*, vol. 132, pp. 2293–2397, 2004.

[29] M. Satoh, "Conservative scheme for the compressible nonhydrostatic models with the horizontally explicit and vertically implicit time integration scheme," *Mon. Weather Rev.*, vol. 130, pp. 1227–1245, 2002.

[30] U. M. Ascher, S. J. Ruuth, and R. J. Spiteri, "Implicit-explicit Runge-Kutta methods for time-dependent partial differential equations," *Appl. Numer. Math.*, vol. 25, no. 2-3, pp. 151–167, Nov. 1997.

[31] A. Steyer, C. J. Vogl, M. Taylor, and O. Guba, "Efficient IMEX Runge-Kutta methods for nonhydrostatic dynamics," *arXiv e-prints*, p. arXiv:1906.07219, Jun 2019.

[32] Top500 supercomputer sites. [Online]. Available: https://top500.org

[33] H. Yashiro, M. Terai, R. Yoshida, S. Iga, K. Minami, and H. Tomita, "Performance analysis and optimization of Nonhydrostatic ICosahedral Atmospheric Model (NICAM) on the K computer and TSUBAME2.5," in *Proceedings of the Platform for Advanced Scientific Computing Conference PASC '16*. ACM Press, 2016. [Online]. Available: https://doi.org/10.1145/2929908.2929911

[34] C. Yang *et al.*, "10M-Core scalable fully-implicit solver for nonhydrostatic atmospheric dynamics," in *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov 2016, pp. 57–68.

[35] H. Fu *et al.*, "Redesigning CAM-SE for peta-scale climate modeling performance and ultra-high resolution on Sunway TaihuLight," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '17. New York, NY, USA: ACM, 2017, pp. 1:1–1:12.

[36] J. Whitaker. (2016) HIWPP non-hydrostatic dynamical core tests: Results from idealized test cases. [Online]. Available: https://www.weather.gov/media/sti/nggps/HIWPP_idealized_tests-v8\%20revised\%2005212015.pdf

[37] J. Michalakes *et al.* (2016) AVEC report: NGGPS level-1 benchmarks and software evaluation. [Online]. Available: https://repository.library.noaa.gov/view/noaa/18654

[38] M. Rančić, R. J. Purser, D. Jović, R. Vasic, and T. Black, "A nonhydrostatic multiscale model on the uniform jacobian cubed sphere," *Monthly Weather Review*, vol. 145, no. 3, pp. 1083–1105, Mar. 2017.

[39] O. Fuhrer *et al.*, “Near-global climate simulation at 1 km resolution: establishing a performance baseline on 4888 GPUs with COSMO 5.0,” *Geosci. Model Dev.*, vol. 11, no. 4, pp. 1665–1681, 2018.

[40] P. H. Lauritzen, C. Jablonowski, M. A. Taylor, and R. D. Nair., Eds., *Numerical Techniques for Global Atmospheric Models*, ser. Lecture Notes in Computational Science and Engineeering. Berlin, Heidelberg, New York: Springer, 2011, vol. 80.

[41] R. D. Hornung and J. A. Keasler, “The RAJA portability layer: Overview and status,” Lawrence Livermore National Laboratory (LLNL), Livermore, CA, Tech. Rep., 2014.

[42] L. V. Kale, E. Bohm, C. L. Mendes, T. Wilmarth, and G. Zheng, “Programming petascale applications with Charm++ and AMPI,” in *Petascale Computing: Algorithms and Applications*, 2008.

[43] D. Medina, A. St-Cyr, and T. Warburton, “OCCA: A unified approach to multi-threading languages,” *SIAM J. Sci. Comput.*, 03 2014.

[44] M. Harris. (2015) Developing portable CUDA C/C++ code with Hemi. [Online]. Available: http://devblogs.nvidia.com/parallelforall/developing-portable-cuda-cc-code-hemi/

[45] H. Kaiser, M. Brodowicz, and T. Sterling, “ParalleX: An advanced parallel execution model for scaling-impaired applications,” in *Proceedings of the 2009 International Conference on Parallel Processing Workshops*, 2009.

[46] (2019) The OpenACC application programming interface version 3.0. [Online]. Available: https://www.openacc.org/sites/default/files/inline-images/Specification/OpenACC.3.0.pdf

[47] (2018) OpenMP application programming interface version 5.0. [Online]. Available: https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5.0.pdf

[48] V. Clement, S. Ferrachat, O. Fuhrer, X. Lapillonne, C. E. Osuna, R. Pincus, J. Rood, and W. Sawyer, “The CLAW DSL: Abstractions for performance portable weather and climate models,” in *Proceedings of the Platform for Advanced Scientific Computing Conference, PASC '18*. ACM, 2018. [Online]. Available: http://doi.acm.org/10.1145/3218176.3218226

[49] E. Anderson *et al.*, *LAPACK Users’ Guide*, 3rd ed. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1999.

[50] Y. Zhang, J. Cohen, and J. D. Owens, “Fast tridiagonal solvers on the GPU,” *ACM Sigplan Notices*, vol. 45, no. 5, pp. 127–136, 2010.

[51] J. Rosinksi. (2017) GPTL – general purpose timing library. [Online]. Available: https://jmrosinski.github.io/GPTL

[52] Kokkos profiling tools. [Online]. Available: https://github.com/kokkos/kokkos-tools

[53] S. Williams, A. Waterman, and D. Patterson, “Roofline: An Insightful Visual Performance Model for Multicore Architectures,” *Commun. ACM*, vol. 52, no. 4, 2009.

[54] C. Yang, T. Kurth, and S. Williams, “Hierarchical Roofline analysis for GPUs: Accelerating performance optimization for the NERSC-9 Perlmutter system.” *Concurr. Comp. Pract. E.*, 2019.

[55] S. S. Vazhkudai *et al.*, “The design, deployment, and evaluation of the CORAL pre-exascale systems,” in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, ser. SC ’18. IEEE Press, 2018. [Online]. Available: https://doi.org/10.1109/SC.2018.00055

[56] SUMMIT Oak Ridge National Laboratory’s 200 petaflop supercomputer. [Online]. Available: https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/

[57] C. Jablonowski and D. L. Williamson, “A baroclinic instability test case for atmospheric model dynamical cores,” *Quarterly Journal of the Royal Meteorological Society*, vol. 132, no. 621C, pp. 2943–2975, 2006. [Online]. Available: https://rmets.onlinelibrary.wiley.com/doi/abs/10.1256/qj.06.12

[58] Dynamical core model intercomparison project. [Online]. Available: https://www.earthsystemcog.org/projects/dcmip-2012/

[59] C. Jablonowski and D. L. Williamson, “A baroclinic instability test case for atmospheric model dynamical cores,” *Q. J. R. Meteorol. Soc.*, vol. 132, pp. 2943–2975, 2006.