Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-218
Manuscript under review for journal Geosci. Model Dev.
Discussion started: 26 October 2018

Geoscientific Model Development Discussions

HOMMEXX 1.0: A Performance Portable Atmospheric Dynamical Core for the Energy Exascale Earth System Model

Luca Bertagna[1], Michael Deakin[1], Oksana Guba[1], Daniel Sunderland[1], Andrew M. Bradley[1], Irina K. Tezaur[1], Mark A. Taylor[1], and Andrew G. Salinger[1]

[1]Sandia National Laboratories, PO Box 5800, Albuquerque, NM, 87175 USA

Correspondence: Luca Bertagna (lbertag@sandia.gov)

Abstract. We present an architecture-portable and performant implementation of the atmospheric dynamical core (HOMME) of the Energy Exascale Earth System Model (E3SM). The original Fortran implementation is highly performant and scalable on conventional architectures using MPI and OpenMP. We rewrite the model in C++ and use the Kokkos library to express on-node parallelism in a largely architecture-independent implementation. Kokkos provides an abstraction of a compute node or device, layout-polymorphic multidimensional arrays, and parallel execution constructs. The new implementation achieves the same or better performance on conventional multicore computers and is portable to GPUs. We present performance data for the original and new implementations on multiple platforms, on up to 5400 compute nodes, and study several aspects of the single- and multi-node performance characteristics of the new implementation on conventional CPU, Intel Xeon Phi Knights Landing, and Nvidia V100 GPU.

1 Introduction

In this paper, we present the results of an effort to rewrite HOMME, a Fortran-based code for global atmosphere dynamics and transport, to a performance portable implementation in C++ (which we will call HOMMEXX), using the Kokkos library and programming model (Carter Edwards et al., 2014) for on-node parallelism. Our definition of performance portable software, for the purposes of this paper, is a single code base that can achieve performance on par with the Fortran implementation on conventional CPU and Intel Xeon Phi Knights Landing (KNL) HPC architectures while also achieving good performance on an Nvidia V100 GPU architecture.

High-Order Methods Modeling Environment (HOMME) is part of the Energy Exascale Earth System Model (E3SM) (E3S), a globally coupled model funded by the United States Department of Energy (DOE), with version 1 released in Spring 2018; HOMME is also part of the Community Earth System Model (CESM) (Hurrell et al., 2013). The development of E3SM is

Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-218
Manuscript under review for journal Geosci. Model Dev.
Discussion started: 26 October 2018

Geoscientific Model Development Discussions
Open Access
EGU

crucial for advancing climate science in directions that support DOE mission drivers over the next several decades, which generally involve energy, water, and national security issues.

The E3SM model is a multiphysics application consisting of many coupled components, including models for the atmosphere dynamics and transport (HOMME), several atmospheric physics sub-grid processes, radiative heat transfer, ocean dynamics and species transport, land surface processes, sea ice dynamics, river runoff, and land ice evolution. The need to resolve numerous physical processes over a broad range of spatial and temporal scales clearly makes E3SM an application that could benefit from effective use of exascale computing resources.

Our project was created to explore, within E3SM, the feasibility of the approach of using C++ (and Kokkos, in particular) to achieve performance on the newest HPC architectures at DOE facilities and to add agility to be prepared for subsequent changes in HPC architectures. We decided to focus on HOMME for several reasons. First, and most important, HOMME represents one of the most critical components of E3SM, typically accounting for 20-25% of run time of a fully-coupled global climate simulation. Second, it is an attractive target for refactoring, as it has a well-defined test suite. Finally, HOMME offers a high bar for comparison, as it has highly-optimized implementations for MPI and OpenMP parallelism that we could use as targets for our benchmarks (Worley et al., 2011).

Our performance improvement efforts were successful in reaching a performance portable code base. In the strong-scaled end of our studies, with relatively fewer spectral elements assigned to each node, HOMMEXX achieves better than performance parity with HOMME on conventional CPU and Intel Xeon Phi KNL, and is faster on a single Nvidia V100 GPU than it is on a single dual-socket, 32-core Intel Haswell (HSW) node. In high-workload regimes where we assign more spectral elements to a node, performance on KNL and GPU is better still: here, HOMMEXX is approximately $1.25\times$ faster than HOMME on KNL, and it is $1.2\times$ to $3.8\times$ faster on a V100 than on a HSW node. In short, HOMMEXX demonstrates that, using C++ and Kokkos, we were able to write a single code base that matched or exceeded the performance of the highly-optimized Fortran code on CPU and KNL while also achieving good performance on GPU. The source code is publicly available at (Bertagna et al., 2018).

As part of the rewrite we also translated the MPI communication layers to C++, adapted them to our layouts, and made minor improvements. Scaling studies show that we did not degrade this capability from HOMME: our largest calculation was a problem of 9.8 Billion unknowns solved on 5400 nodes and 345600 ranks on Cori-KNL, and was marginally faster with HOMMEXX than HOMME.

The path to reaching this result required effort. Our initial code translation of HOMME into C++ and Kokkos was indeed portable to all architectures; however, by having the original implementation to compare against, we knew that this early version was not at acceptable performance levels. The performance of HOMMEXX as presented in this paper was eventually achieved by carefully studying the most computationally-demanding routines and optimizing the code, e.g., by reducing memory movement and using explicit vectorization where possible.

Recent years have seen a number of efforts to create performance portable versions of existing libraries and codes, most of which have focused on portability to GPUs. Here, we mention a few of these endeavors that share some common aspects with our work.

Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-218
Manuscript under review for journal Geosci. Model Dev.
Discussion started: 26 October 2018

In (Carpenter et al., 2013) the authors present an effort to port a key part of the tracer advection routines of HOMME to GPU using CUDA Fortran, with good success, while in (Norman et al., 2015) a similar effort is described, but this time using OpenACC, and comparing results with a CUDA Fortran implementation, as well as a CPU implementation. The results were analyzed in terms of timings, development effort, and portability of the loops structure to other architectures.

In (Fu et al., 2016) the authors describe the performance of a version of the Community Atmosphere Model (CAM) ported to run on Sunway supercomputers, using OpenACC. Their porting effort shows good performance on some key kernels, including kernels for atmospheric physical processes. Reference (Fu et al., 2017) also describe the performance of CAM on Sunway architecture, using both OpenACC and Athread (a lightweight library designed specifically for Sunway processors), on up to 41,000 nodes. In this work, Athread is shown to outperform OpenACC by a factor as high as 50x on some kernels.

In (Spotz et al., 2015; Demeshko et al., 2018), the authors discuss the performance portability of the finite element assembly in Albany (Salinger et al., 2016) (an open-source, C++, multi-physics code) to multi/many-core architectures. Attention was focused on the Aeras global atmosphere model within Albany, which solves equations for atmospheric dynamics similar to those in HOMME (2D shallow water model, 3D hydrostatic model). Numerical results were presented on a single code implementation running across three different multi/many-core architectures: GPU, KNL and HSW.

A third performance-portability effort in the realm of climate involves the acceleration of the Implicit-Explicit (IMEX) Non-hydrostatic Unified Model of the Atmosphere (NUMA) on manycore processors such as GPUs and KNLs (Abdi et al., 2017). Here, the authors utilized the OCCA (Open Concurrent Compute Abstraction) many-core portable parallel threading run-time library (Medina et al., 2014), which offers multiple language support. Strong scalability of their IMEX-based solver was demonstrated on the Titan supercomputer's K20 GPUs, as well as a KNL cluster.

In (Fuhrer et al., 2014), a refactoring effort for the atmospheric model COSMO (Fuhrer et al., 2017) is presented. The authors describe the strategy as well as the tools used to obtain a performance portable code. As for HOMME, the original code is in Fortran, while the refactored code is in C++, and makes extensive use of generic programming and template metaprogramming. A comparison is shown with the original implementation, showing an improvement in the production code performance.

Although our work shares common features with all of the above, it is the combination of several aspects that makes it unique: first, our approach aims to obtain a single implementation, performant on a variety of architectures; second, except for initialization and I/O, our implementation is *end-to-end* on the device (meaning no copies device to host or host to device are needed); third, the original implementation already achieved good performance on CPU and KNL, which our implementation has to be on par with; finally, we rely on Kokkos for portability, leveraging an existing cutting-edge library rather than implementing an in-house new interface.

The remainder of this paper is structured as follows. In section 2 we give an overview of HOMME, its algorithms, and computational capabilities. In section 3 we briefly present the Kokkos library and its main features, and describe the implementation process and choices we used in the development of HOMMEXX. Section 4 is dedicated to an in-depth performance study of HOMMEXX, including strong scaling, on-node performance, and comparison with the original HOMME. Finally, in section 5 we summarize what we have achieved and discuss lessons learned from our porting effort.

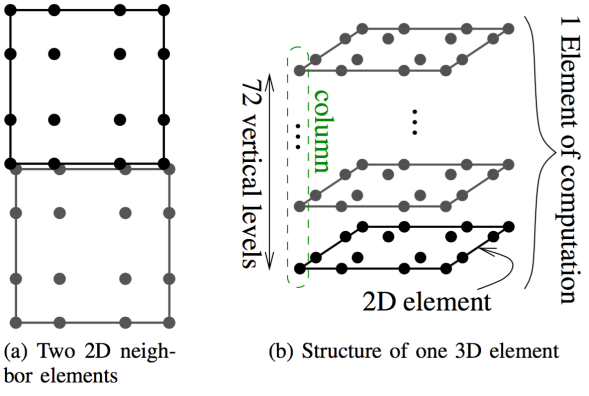
Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-218
Manuscript under review for journal Geosci. Model Dev.
Discussion started: 26 October 2018

Geoscientific Model Development Discussions

2 The HOMME Dycore

Atmospheric dynamics and tracer transport in E3SM are solved by the High-Order Methods Modeling Environment (HOMME) dynamical core. HOMME utilizes the Spectral Element Method (SEM) (Taylor, 2012), which is a member of the continuous Galerkin finite-elements schemes. The SEM is a highly competitive method for fluid dynamics applications, due to its accuracy and scalability (Maday and Patera, 1989; Dennis et al., 2005; Canuto et al., 2007; Bhanot et al., 2008).

In models of global atmospheric circulation, it is common to reformulate the governing Navier-Stokes and transport equations for a new vertical coordinate, and to decouple horizontal (2D) and vertical (1D) differential operators, as described in (Taylor, 2012; Dennis et al., 2012). In HOMME, a horizontal mesh of conforming quadrilateral elements is partitioned between MPI ranks, with each vertical column belonging to only one rank. Vertical operations act on vertical levels and are resolved either with Eulerian or Lagrangian operators; the latter are based on remapping procedures (briefly, “remaps"). For the horizontal direction, differential operators are discretized with the SEM. A typical element structure, including the distribution of the degrees of freedom (DOFs), and vertical coordinate system is depicted in Fig. 1. In HOMME’s version of the SEM, the DOFs and the quadrature points coincide. Figure 2 illustrates a typical quadrilateral mesh based on a cubed-sphere with 10 elements per cube edge ($n_e = 10$). On a cubed-sphere mesh, the total number of elements is $6 \cdot n_e^2$, and the number of DOFs per variable is $6 \cdot n_e^2 \cdot n_p^2 \cdot n_l$, where n_p is the number of GLL points per element edge, and n_l is the number of vertical levels. By default, in E3SM, $n_p = 4$ and $n_l = 72$. For the most refined mesh in this study, with $n_e = 240$, this leads to approximately 398×10^6 DOFs per variable. Figure 4 contains a visualization of total precipitable water obtained from an E3SM simulation with HOMME at $n_e = 240$.



(a) Two 2D neighbor elements

(b) Structure of one 3D element

Figure 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 72 2D elements, each with 16 DOFs (GLL) points.

The dynamical part of HOMME (briefly, “dynamics") in E3SM solves for 4 prognostic variables (pressure, temperature, and two horizontal velocity components), using a 3rd order 5-stage Runge-Kutta (RK) method and a hyperviscosity (HV) operator

Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-218
Manuscript under review for journal Geosci. Model Dev.
Discussion started: 26 October 2018

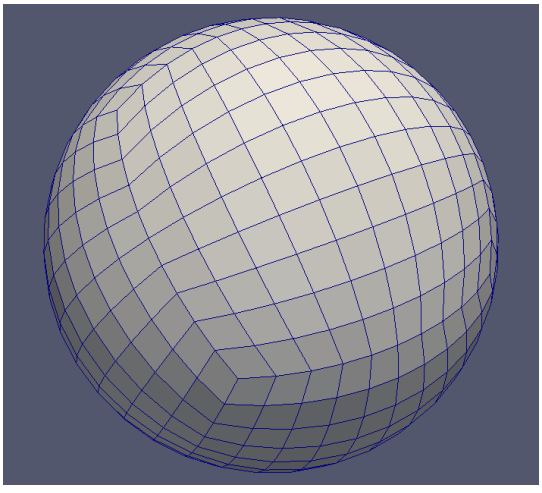


Figure 2. Example of a cubed-sphere quadrilateral mesh with $n_e = 10$, yielding 600 elements.

for grid scale dissipation (Guba et al., 2014b). Due to timestepping restrictions, the hyperviscosity operator is subcycled 3 times per RK step. The tracer transport part (briefly, “tracers") typically solves for 40 prognostic tracers, using a 2nd-order 3-stage RK method, hyperviscosity operator, and a limiting procedure for shape preservation. The limiter is based on a local shape preserving algorithm that solves a quadratic program (Guba et al., 2014a). For vertical operators, we will only consider a Lagrangian formulation, which requires a conservative and shape-preserving remapping algorithm. For the remapping algorithm, we use the piece-wise parabolic method (Colella and Woodward, 1984). A schematic of the operations for dynamics, tracers, hyperviscosity, and remap is shown in Fig. 3 as a code flow chart. Settings and parameters in our simulations match those commonly used in E3SM production runs for 1° degree horizontal resolution ($n_e = 30$).

As usual for finite-elements codes, SEM computations are performed in two stages: during the first stage, on-element local differential operators are computed; during the second stage, each rank exchanges information with its neighbors and performs a weighted averaging for the solution. Local differential operators consist of weak versions of gradient, divergence, curl, and their compositions, and are implemented as matrix-vector products.

As shown in Fig. 3, typical timestepping in HOMME consists of 3 horizontal steps per one vertical remapping step. The horizontal step consists of three sequential parts: 5 RK stages for the dynamics; 3 subcycles of the hyperviscosity application for the dynamics variables; and 3 RK stages for tracers, with each stage carrying its own logic for hyperviscosity, limiter, and MPI exchanges that are required for the limiter. All performance results in this paper use this configuration.

Vertical operators are not computationally intensive and are evaluated locally, while the RK stages and HV subcycles consist mostly of several calls to 2D spherical operators. Therefore, Fig. 3 also highlights that a significant part of a full HOMME time step is spent in MPI communications and on-node 2D spherical operators. Due to a diagonal mass matrix in the SEM, MPI communications involve only edges of the elements. For production runs, with O(10^4)–O(10^5) ranks and 1–2 elements per rank on CPUs, MPI cost becomes dominant relative to that of on-node computations. Previous extensive development

Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-218
Manuscript under review for journal Geosci. Model Dev.
Discussion started: 26 October 2018

Geoscientific Model Development Discussions
Open Access
EGU

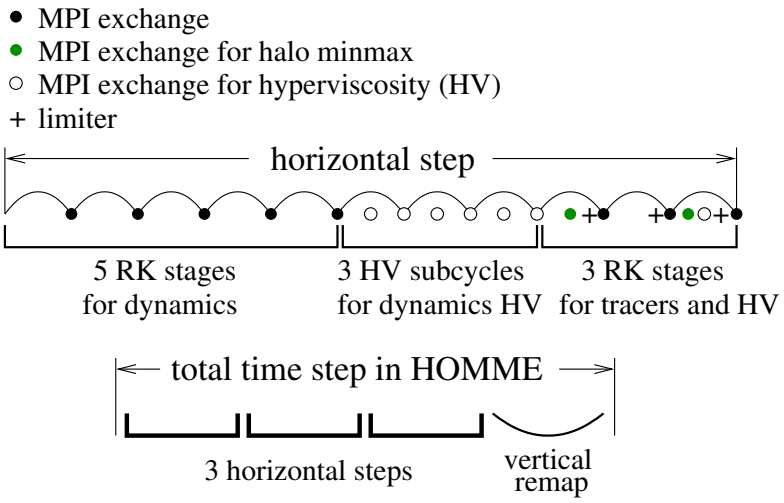


Figure 3. Code flow chart for HOMME timestepping. Parameters match the ones selected for E3SM.

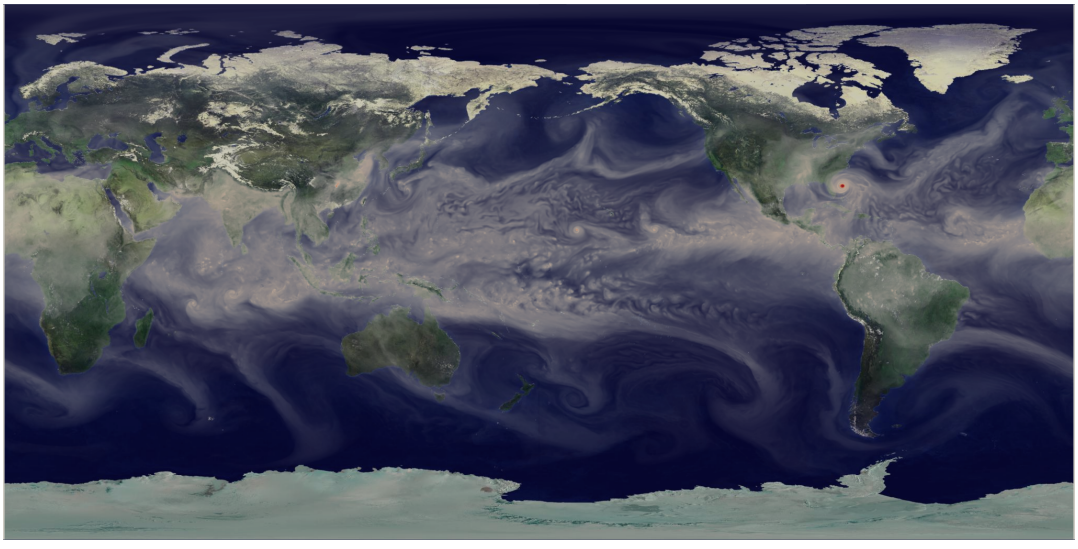


Figure 4. Visualization of total precipitable water, from a 1/8 degree ($n_e = 240$) E3SM simulation. The simulation used active atmosphere, land and ice component models, with prescribed ocean sea surface temperature and prescribed sea ice extent. Dynamics in HOMME alone accounts for roughly 1.6B degrees of freedom.

efforts in HOMME led to very efficient MPI communication procedures. In this work, we adopt the same MPI communication patterns as in HOMME, and focus on performance and portability of on-node computations.

On-node computations can be parallelized through threading. On conventional CPUs, HOMME supports outer OpenMP threading over elements and inner OpenMP threading over vertical levels and tracers. For efficiency, threading over elements

Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-218
Manuscript under review for journal Geosci. Model Dev.
Discussion started: 26 October 2018

is performed in one large parallel region. Each type of threading can be turned on or off at configure time, and, if both are on, they lead to *nested* OpenMP regions. For most of our performance studies we configure HOMME with outer threading only, unless stated otherwise.

3 Overview of the implementation

3.1 The Kokkos library

Our strategy for achieving performance portability of HOMME focuses around Kokkos, a C++11 library and programming model that enables developers to write performance portable thread-parallel codes on a wide variety of HPC architectures (Carter Edwards et al., 2014). Kokkos is used to optimize on-node performance, allowing HPC codes to leverage their existing strategies for optimizing inter-node performance. Here, we briefly describe the key premises on which Kokkos is based.

Kokkos exposes several key compile time abstractions for parallel execution and data management which help its users to develop performance portable code. The following execution abstractions are used to describe the parallel work.

- A **kernel** is a user provided body of work that is to be executed in parallel over a collection of user defined work items. Kernels are required to be free of data dependencies, so that a kernel can be applied to the work items concurrently without an ordering.
- An **execution space** describes where a kernel should execute; for example, whether a kernel should run on the GPU or the CPU.
- An **execution pattern** describes how a kernel should run in parallel. Common execution patterns are `parallel_for`, `parallel_reduce`, and `parallel_scan`.
- The **execution policy** describes how a kernel will receive work items. A **range** execution policy is created with a pair of lower (L) and upper (U) bounds, and will invoke the kernel with an integer argument for all integers i in the interval [L, U). On the other hand, a **team** execution policy is created with a number of teams and number of threads per team. The strength of a team policy is that threads within a team can cooperate to perform shared work, allowing for additional levels of parallelism to be exposed within a kernel. Team policies allow users to specify up to three levels of hierarchical parallelism: over the number of teams, the number of threads in a team, and the vector lanes of a thread.

Kokkos also provides memory abstractions for data. The main one is a multidimensional array reference which Kokkos calls `View`. `Views` have four key abstractions: (1) data type, which specifies the type of data stored in the `View`; (2) layout, which describes how the data is mapped to memory; (3) memory space, which specifies where the data lives; and (4) memory traits, which indicates how the data should be accessed. The `View` abstractions allow developers to code and verify algorithms one time, while still leaving the flexibility in the memory, such as transposing the underlying data layouts for CPU versus GPU architectures.

Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-218
Manuscript under review for journal Geosci. Model Dev.
Discussion started: 26 October 2018

Kokkos uses C++ template metaprogramming to specify the instantiations of the execution space and data/memory abstractions of data objects and parallel executions constructs, to best optimize code for the specified HPC architecture. This allows users to write on-node parallel code that is portable and can achieve high performance.

HOMMEXX relies on Kokkos `Views` for data management, while parallelization of work is achieved through calls to the `parallel_for`, `parallel_reduce`, and `parallel_scan` templated functions.

We point out that, while we chose to use Kokkos to achieve performance portability, other performance-portable programming models exist, e.g., RAJA (Hornung and Keasler, 2014), HEMI (Harris, 2015), OCCA (Medina et al., 2014), Charm++ (Kale et al., 2008) and ParalleX (Kaiser et al., 2009). These models are compared and contrasted with Kokkos in (Demeshko et al., 2018). OpenACC (Ope, 2017) is another performance-portable model, but has limited compiler support. Kokkos's parallel constructs are also similar to future STL parallel extensions (par).

3.2 Conversion of the existing HOMME Fortran library to C++ and Kokkos

HOMMEXX is an incremental conversion of an existing Fortran 90 code to a C++ code which utilizes the Kokkos programming model for on-node parallelism. To ensure correctness of the new code base, we set up a suite of test cases for comparing the solution computed using HOMMEXX with the solution obtained using the original HOMME. To simplify this comparison, we enforced a *bit-for-bit* match between the solutions in correctness-testing builds.

For tests, compiler optimizations and fused multiply-add are disabled using `-fmad=false` for CUDA, `-fpmodel=strict` for Intel, and `-ffp-contract=off` for GNU. For the GPU, team reductions are serialized using a configuration option.

Once the testing framework was set up, we proceeded with the conversion effort. In particular, the code was divided into six primary algorithm sets:

- differential operators on the sphere;
- an RK stage for the dynamics;
- an RK stage for the tracers, including MPI, hyperviscosity, and limiter;
- application of hyperviscosity to the dynamics states;
- vertical remap of states and tracers;
- exchange of quantities across element boundaries.

Details on the implementation of these algorithms will be given in section 3.4.

The single RK stage was the first part of the code to be converted. Initially, we separated it from the rest of the code to perform a study on the possible design choices, e.g., data layout, parallelism, and vectorization. Once the design choices were made, we continued the conversion of the kernels, one by one, while maintaining bit-for-bit agreement with the original HOMME. We kept the original Fortran implementation of the main entry point, as well as initialization and I/O, for the duration of the conversion process. During development, copies of data were required between C++ and Fortran code sections because the two,

Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-218
Manuscript under review for journal Geosci. Model Dev.
Discussion started: 26 October 2018

for performance reasons, use different memory layouts; see the next sections for details. As the conversion process progressed, we were able to push the C++ code entry point up the stack, until we were able to avoid any data movement between Fortran and C++ except in I/O and diagnostic functions.

3.3 Performance and optimization choices

Performance factors depend on architecture. For conventional CPU with DDR4, minimizing memory movement is the most important; for KNL, keeping a workset in high-bandwidth memory and vectorization are; for GPU, maximizing parallelism is. A performance portable design must account for all of these in one implementation.

As we mentioned in the introduction, the initial translation of HOMME into C++ with Kokkos was not as performant as the original Fortran code. This was not surprising, considering that the Fortran code had already gone through several stages of optimization. Furthermore, it must be recalled that Kokkos is not a black box library that automatically makes an existing code performant, and effort must be put into writing efficient parallel code. In the case of HOMMEXX, we performed several optimizations, that gradually improved performance. Among the different optimization choices, we will now discuss the three that gave the largest performance boost (in the range of 10%-50% speedup); several other micro-optimizations (such as temporaries removal or loops reorganization) also proved important, though individually they usually accounted for only a few percent points speedups. We point out that the speedup given by a particular optimization choice is only relative to the quality of the code before such optimization is introduced, so that the the same choice may prove more or less effective at a different development stage or on a different code. Nevertheless, we believe the following three design choices were indeed crucial for HOMMEXX to achieve performance, and, adapted to the context, are likely to be relevant also for different applications.

Our approach for optimizing performance centered on exploiting hierarchical parallelism. HOMME has many nested loops, over elements, GLL points within an element, and vertical levels within an element. In HOMMEXX, these nested loops are parallelized using team policies, described in section 3.1. Since, within a kernel, each element is completely independent from the others, the outer `parallel_for` is over the elements in the MPI rank. If the kernel has to be run for several variables, as in the tracers RK stage and vertical remap, the outer `parallel_for` is dispatched over the variables in all the elements. The `parallel_for` for teams and vector lanes are dispatched over the GLL points within an element and the vertical levels, respectively. This hierarchical structure exposes parallelism with minimal bookkeeping and enables team-level synchronization. It is particularly natural for the horizontal differential operators. Such operators couple DOFs within an element. Exposing fine-grained parallelism with a simple range policy would require substantial effort. In a team policy approach, a team collectively computes a differential operator, synchronizes, and then each member uses part of the result for further calculations.

The next important design choice focused simultaneously on vectorization on CPU and KNL architectures and efficient threading and coalesced memory access on GPU. The two most natural choices for the primary vectorization direction are the vertical levels and the GLL points. In the former case, Kokkos' thread and vector level of parallelism would be on GLL points and vertical levels respectively, while in the latter it would be the opposite. Our choice in HOMMEXX was to go with the former, because of the following advantages compared to the latter: first, it trades vectorizing in the vertical summation regions

Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-218
Manuscript under review for journal Geosci. Model Dev.
Discussion started: 26 October 2018

Geoscientific Model Development Discussions

for vectorizing in the much more frequent spherical operators; second, Kokkos' hierarchical parallelism currently offers a parallel scan implementation for vector level operations, but not for thread level operations, which is relevant for vertical summations; finally, while it is reasonable to change the number of levels to reach multiples of the vector size, doing so for the number of points (which is a perfect square) may quickly become computationally intractable as vector sizes grow.

There are several benchmarks to measure how successfully a compiler can vectorize code; see, e.g., (Rajan et al., 2015; Callahan et al., 1988). In HOMMEXX, we use explicit vectorization, motivated by the observation that the C++ code was not benefiting enough from the compiler's automatic vectorization. In contrast, the Fortran compiler appears to be more successful in generating vectorized code, likely as a result of the language's built-in array arithmetic (Intel, R).

To achieve vectorization, we replaced the core data type, a single `double`, with a templated structure, `Vector`, wrapping an array of N `doubles`. Here, N is a template parameter of the class `Vector`, which is selected at configure time depending on the available vector-unit length on the current architecture (e.g., $N = 4$ on HSW, $N = 8$ on KNL). This structure provides overloads of basic arithmetic operations (+,-,*,/). Each overload calls the corresponding vector intrinsic (Intel, R). We observed that explicit vectorization provides a 1.15–1.2$\times$ speedup on KNL relative to simply blocking the loops and applying `#pragma simd`. On GPUs, however, since vector parallelism is achieved through SIMT rather than SIMD, this abstraction doesn't improve performance, so we use the `Vector` wrapper with $N = 1$. A templated structure to enforce explicit vectorization is not an original concept: other efforts have been made, e.g. (Estérie et al., 2012), and any such structure would have served our purposes.

The use of `Vector` on CPU and KNL is efficient throughout most of the code: operations proceed independently of level, and there is no inter-level dependency. However, there are a handful of routines where at least one of these conditions does not hold. In both vertical remap and the tracer mixing ratio limiter, algorithms have a high density of conditional statements, which make vectorization inefficient. In addition, several computations require a scan over the levels. For these, we isolate the scan from the rest of the operations to minimize non-SIMD execution.

The third important design choice concerned memory movement. Reducing memory movement is primarily important on conventional CPUs, which have less bandwidth than new architectures, but it improves performance on all architectures. For transient intermediate quantities, shared, reused workspace that has size proportional to number of threads is used. Persistent intermediate quantities are minimized. This proved especially important in the implementation of the horizontal differential operators (see section 3.4.2).

3.4 Implementation details

In this section, we highlight a few implementation details.

3.4.1 MPI

HOMMEXX uses the same MPI communication pattern as HOMME. In fact, the connectivity pattern between elements is computed with the original Fortran routines, and then forwarded to the C++ code. All communications are nonblocking and two-sided. The minimal number of messages per communication round is used. Each rank packs the values on the element

Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-218
Manuscript under review for journal Geosci. Model Dev.
Discussion started: 26 October 2018

halo in a send buffer; posts nonblocking MPI sends and receives; and unpacks the values in a receive buffer, combining them with the local data.

In HOMMEXX, this process is handled by the `BoundaryExchange` class. Several instances of this class are present, each handling the exchange of a different set of variables. Internally, these instances use, but do not own, two raw buffers: one for exchanges between elements on process (not involving MPI calls), and one for exchanges between elements across processes (requiring MPI calls). The buffers are created and handled by a single instance of a `BuffersManager` class, and are shared by the different `BoundaryExchange` object, reducing the required buffer space.

The implementation of the pack and unpack kernels is one of the few portions of HOMMEXX that differ for GPU and CPU/KNL architectures. On CPU and KNL platforms, reuse of subviews is important to minimize index arithmetic, which motivated the choice of parallelizing only on the number of local elements and exchanged variables. On GPUs, maximizing occupancy is important, which motivated the choice of parallelizing the work not only over the number of elements and exchanged variables, but also on the number of connections per element and the number of vertical levels, at the cost of more index arithmetic. Although this choice does not align with the idea of a single code base, we point out that architecture-dependent code is a very limited portion of HOMMEXX.

Since CUDA-aware MPI implementations support the use of pointers on the device, there is no need for application-side host-device copies of the pack and unpack buffers for GPU builds. This can be especially beneficial if combined with GPU-Direct or NVLink technologies for fast GPU-GPU and GPU-CPU communication (NVIDIA, C).

3.4.2 Differential operators on the sphere

Discrete differential operators on the sphere are defined only in the horizontal direction, and include gradient, divergence, curl, and Laplacian. Routines implementing these operators are called extensively throughout the code, such as the RK stages for dynamics and tracers and the hyperviscosity step. In the original Fortran implementation, such routines operate on an individual vertical level. The action of an operator on a 3D field is then computed by subviewing the field at each level, and passing the subview to the proper routine. As mentioned in section 2, HOMME can only thread over elements and vertical levels (or tracers), but has no support for threading over GLL points. Therefore, the body of each differential operator is serial, optimized only by means of compiler directives to unroll small loops and to vectorize array operations.

In HOMMEXX, we implemented all the differential operators on the sphere with the goal to expose as much parallelism as possible. As mentioned in section 3.3, HOMMEXX uses a three-layers hierarchical parallelism, with `parallel_for` loops over elements (possibly times the number of tracers), GLL points and vertical levels. The vectorization choice described in section 3.3 makes it natural to proceed in batch over levels. In our implementation, all the differential operators are computed over a whole column. Therefore, on CPU and KNL, operations are vectorized across levels, and on GPU, memory access is coalesced across levels. Some differential operators also require temporary variables, for instance, to convert a field on the sphere to a contravariant field on the cube, or vice versa. Since allocating temporaries at every function call is not performant, the `SphereOperators` class allocates its buffers at construction time. As described in section 3.3, the buffers are only large enough to accommodate all the concurrent teams.

Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-218
Manuscript under review for journal Geosci. Model Dev.
Discussion started: 26 October 2018

4 Performance results

Performance data were collected on a number of platforms and architectures. All computations were done in double precision. In this section, we first provide details of the platforms and architectures and discuss power consumption. Then we describe results for four different data collection types: strong scaling and comparison at scale, strong scaling at small scale but in more detail, single-node or device performance, and GPU kernel performance.

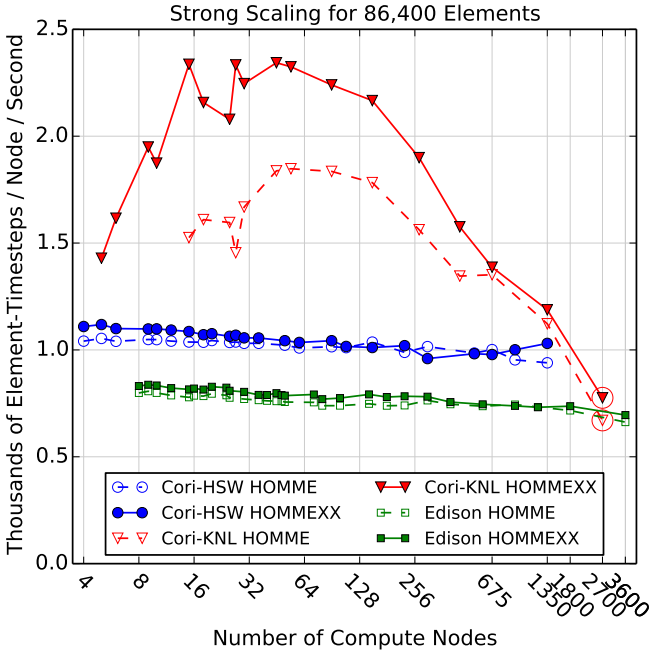


Figure 5. Strong scaling at large scale on NERSC platforms for grid generated with $n_e = 120$ as in Fig. 2. The number, in thousands, of elements that can be integrated one total time step (as shown in Fig. 3), per node, per second, is plotted as a function of the number of nodes. Each of the $86,400$ elements represents the work for the fourth-order spectral element extruded in 3D for 72 vertical levels, as shown in Fig. 1(b), for dynamics and also convection of 40 tracers. Higher values indicate better performance. A horizontal line represents ideal strong scaling. Red circles indicate threads are used within an element.

4.1 Platforms and architectures

First we describe each architecture and platform. Details of particular interest pertain to on-node parallelism, vectorization, memory bandwidth, and power consumption.

The NERSC Cori KNL partition has 54 cabinets and 9688 compute nodes. Each node is a 68-core Intel Xeon Phi Knights Landing 7250 processor, with thermal design power (TDP) of 230 Watts (W). Each core has 4 hardware threads and 2 512-bit-wide vector processing units (VPU). Memory consists of 16 GB MCDRAM with approximately 460 GB/s peak bandwidth and 96 GB DDR4 2400 MHz memory with approximately 102 GB/s peak bandwidth. All runs used the quadrant NUMA mode and the cache memory mode.

Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-218
Manuscript under review for journal Geosci. Model Dev.
Discussion started: 26 October 2018

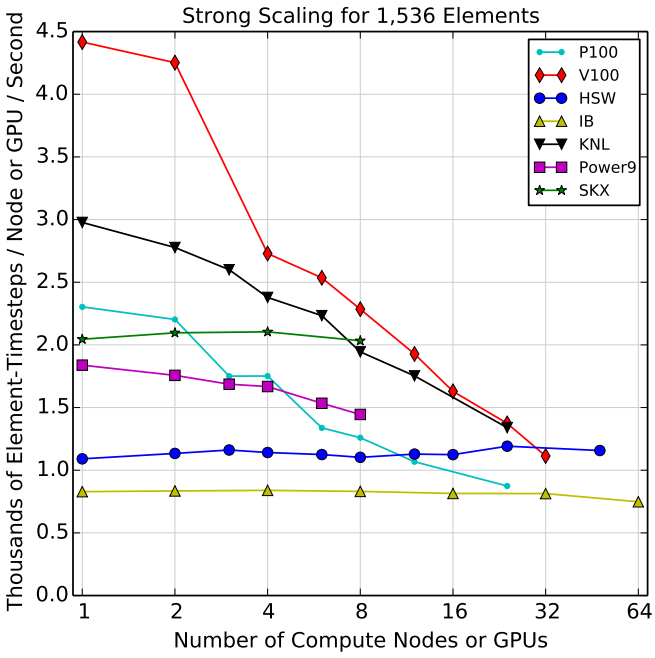


Figure 6. Strong scaling at small scale with Intel Xeon Haswell (HSW), Ivy Bridge (IB), and Skylake (SKX); Intel Xeon Phi Knights Landing (KNL); IBM Power9; and Nvidia GPU Pascal (P100) and Volta (V100) architectures for a $1,536$-element grid generated with $n_e = 16$. See caption of Fig. 5 for details.

The NERSC Cori Haswell (HSW) partition has 14 cabinets and 2388 nodes. A node has 2 sockets. A socket is a 16-core Intel Xeon E5-2698 v3 Haswell, with 165W TDP. Each core has 2 hardware threads and 2 256-bit-wide VPU. Each socket has a 40 MB L3 cache. Each node has 128 GB DDR4 2133 MHz memory.

The NERSC Edison platform has 30 cabinets and 5586 nodes. Each node has 2 sockets. A socket is a 12-core Intel Xeon E5-2695 v2 Ivy Bridge (IB) processor, with 115W TDP. Each core has 2 hardware threads and 1 256-bit-wide VPU. Each socket has a 30 MB L3 cache. Each node has 64 GB DDR3 1866 MHz memory.

Each node of the Nvidia Volta GV100 testbed has a dual-socket POWER9 processor. Each socket has 10 cores, each with 4 threads, and 2 GPUs. We were unable to find the POWER9 TDP. Each GV100 has 2560 double-precision cores, with 32 threads/core, and has a 300W TDP. The GCC 7.20 compiler with CUDA 9.1.85 were used. In GPU runs, one MPI rank was assigned to each GPU. We also obtained data on the older Nvidia Pascal GP100. Each node of the Pascal testbed has a dual-socket POWER8 Firestone processor. Each socket has 8 cores and 2 GPUs. Each GP100 has 1792 double-precision cores, with 32 threads/core, and has a 300W TDP. The GCC 5.4.0 compiler with CUDA 8.0.44 were used.

Finally, each node of the Intel Xeon Skylake (SKX) testbed has 2 Intel Xeon Platinum 8160 Skylake sockets, each with 24 cores, and each with a 150W TDP. Each core has 2 hardware threads and 2 512-bit-wide vector processing units. Each socket has a 32 MB L3 cache. Each node has 196 GB DDR4 2666 MHz memory. Importantly, in addition to having faster memory relative to HSW, SKX also has 6 memory channels relative to HSW's 4.

Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-218
Manuscript under review for journal Geosci. Model Dev.
Discussion started: 26 October 2018

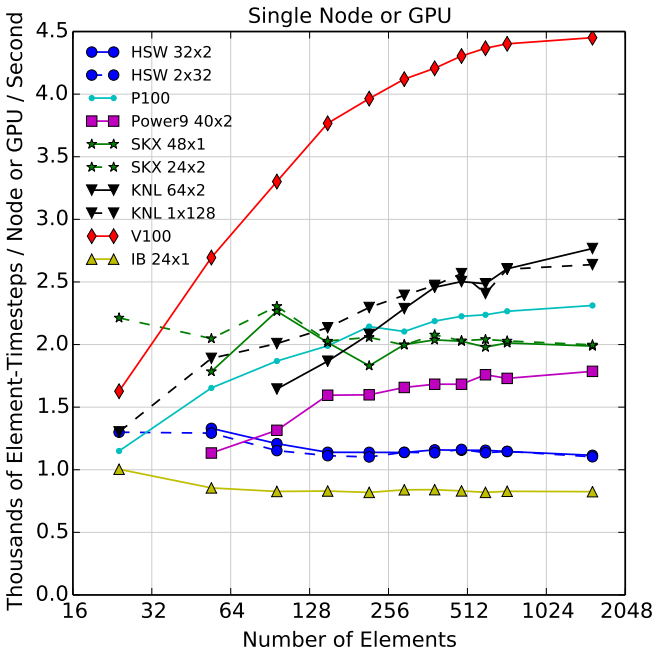


Figure 7. Single-node performance as a function of number of elements for several architectures, with various threading options. See caption of Fig. 6 for architecture names. Number of MPI ranks r and number of threads t in a configuration is denoted $r \times t$. Data are for HOMMEXX only.

Power consumption is an important consideration in new architectures. First we describe power consumption data; then we discuss how these data may be used to provide one perspective for Figs. 5–8. Above, we provided manufacturer-specified thermal design power for each socket or device; we were unable to find data for POWER8 and POWER9. Node-level consumption is complicated by other consumers of power. For example, following (Grant et al., 2017), the maximum power per node is for HSW 415W and for KNL 345W. Without frequency throttling, a HSW node was observed in (Grant et al., 2017) to run at approximately 310 to 355W; without frequency throttling, a KNL node was observed to run at 250–270W. Thus, we conclude that 30W should be added to each CPU TDP to account for other on-node power consumption. The TDP for both the P100 and V100 is 300W, but we observed by continuous sampling by `nvidia-smi` on a single-GPU run of HOMMEXX that the V100 runs at approximately 200W for a HOMMEXX workload that fully occupies the GPU. The host processor's power must be considered; in the absence of data, we assign the POWER9 node to have 360W.

In consideration of these complexities, we offer the following numbers as rough guidelines. An IB node consumes 260W; HSW, 360W; SKX, 330W; KNL, 260W. On Summit, each node will have 6 V100s; thus, we divide the host consumption of 360W among the 6 V100s for a total of 260W per V100. These numbers are too uncertain to provide a useful abscissa in our plots; however, the reader may use these numbers to estimate efficiency in terms of power consumption.

Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-218
Manuscript under review for journal Geosci. Model Dev.
Discussion started: 26 October 2018

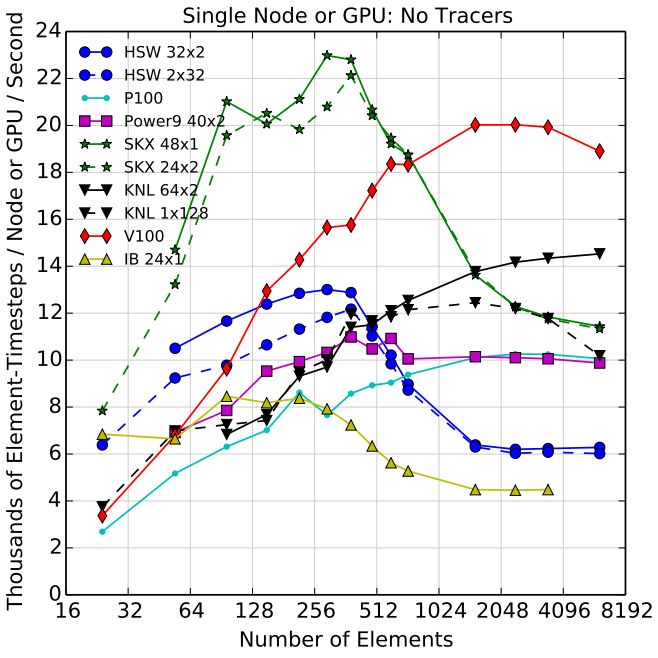


Figure 8. As Fig. 7, but with no tracers.

In Figs. 5 and 6, the abscissa is the number of compute nodes or devices. On GPU-enabled systems, there is more than one GPU per compute node. In this case, we use the smallest number of nodes able to accommodate the given number of GPUs and count the number of GPUs rather than nodes. In Figs. 7 and 8, the abscissa is the number of 3D elements.

The ordinate of these figures is thousands of 3D elements $\times$ total time steps computed per second, per node or GPU, where a total time step is as illustrated in Fig. 3. A 3D element is the full 3D extruded domain of 72 levels as shown in Fig. 1. Hence, the ordinate expresses computational efficiency; larger ordinate values indicate better performance. Efficiency may be expressed in terms of approximate power consumption rather than per node or device by dividing an efficiency value by the architecture's node power value; this calculation gives thousands of elements $\times$ time steps per Joule, i.e., number of computations per unit of energy.

All single-thread runs were done with a serial—i.e., without OpenMP—build. The HOMME and HOMMEXX runs for a particular comparison configuration were done in the same batch allocation.

4.2 Strong scaling and comparison at large scale

Figure 5 reports the performance results of large runs on the NERSC Cori Haswell partition (Cori-HSW), Cori KNL partition (Cori-KNL), and Edison, with 86,400 elements. (We expect to add results from the new Summit supercomputer using NVIDIA Volta GPU devices once we have access to that machine.) On Cori-HSW and Edison, runs were with 1 thread, 1 rank/core. On Cori-KNL, runs were with 1 rank/core, 2 threads/core, 64 ranks/node, except for the red-circled points. These configurations are used in E3SM and thus are the best for a performance comparison study.

Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-218
Manuscript under review for journal Geosci. Model Dev.
Discussion started: 26 October 2018

After these data were collected, we discovered that HOMME was running a loop that adds zeros to the tracers, as part of source term tendencies that are active only when HOMME is coupled to physics parameterizations. These loops account for approximately 1.5% of the raw HOMME time. To compensate for this discrepancy, in Fig. 5, we removed the maximum time over all ranks for this loop from the reported time. This is a conservative fix in that the timing error favors HOMME rather than HOMMEXX; at worst, too much time is subtracted from HOMME's overall time rather than too little.

The red-circled points in Fig. 5 were run with 1 thread/core but 2 cores/element. Runs were carried out to 1 element/core; in the case of the red-circled points, extra runs were carried out to 2 cores/element. In this figure, ideal scaling corresponds to a horizontal line. A run was conducted at 345,600 elements ($n_e = 240$) on up to 5400 Cori KNL nodes. Results are essentially the same as in Fig. 5: HOMMEXX's peak efficiency is 2.5, and efficiency drops to 1.2 at 500 nodes.

The roughly parabolic shape of the KNL curves is the result of exhausting high-bandwidth memory (HBM) at small node count, and a combination of MPI dominance and on-node efficiency loss at low workload at large node count. Regarding efficiency loss at low workload, see also Fig. 7. With $n_q = 40$ tracers, $n_p = 4$, $n_l = 72$ levels, roughly 6 thousand elements fit in the approximately 16 GB of HBM available to the application on a KNL node.

4.3 Strong scaling at small scale in more detail

Figure 6 studies strong scaling at small scale, with number of elements 1,536, with the same configuration as in Section 4.2 but across a broader range of architectures. Here, only HOMMEXX is studied. Again, ideal scaling corresponds to a horizontal line. In both Figs. 5 and 6, Intel Xeon efficiency is roughly flat with increasing node count. In contrast, KNL, GPU, and possibly POWER9 efficiency decreases with node count. The effects of the two sockets on GPU performance (transition from 2 to 3 devices) and then multiple nodes (transition from 4 to 5 devices) are particularly evident. We anticipate internode communication will be better on Summit than on our testbed.

4.4 Single node or device performance

Figures 7 and 8 show results for single-node runs, with 40 tracers as before and also without tracers, respectively. While tracers will always be included in a simulation, far fewer than 40 may be used in some configurations. In addition, a simulation without tracers exposes performance characteristics of the substantially longer and more complicated non-tracer sections of code, as well as of the architectures running that code. The no-tracer configuration may be a useful proxy for applications that do not have the benefit of a small section of code devoted to computation over a large number of degrees of freedom.

In the legend, $r \times t$ denotes r ranks with t threads/rank. With tracers, the key result is that on a single node, performance is roughly independent of the separate factors r, t, depending only on the product $r \cdot t$. Without tracers, a large number of threads per rank does not always perform as well, likely because arrays are over elements, and so two arrays used in the same element-level calculation might have data residing on different memory pages.

The runs without tracers (Fig. 8) on Intel Xeon CPUs, and also evident to a lesser extent in Fig. 7, show that CPU efficiency can vary by as much as a factor of 2 based on workload on a node. This efficiency variation is likely due to the L3 cache. Without tracers, approximately 300 elements can be run on each HSW socket and remain completely in L3. This estimate,

Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-218
Manuscript under review for journal Geosci. Model Dev.
Discussion started: 26 October 2018

doubled because a node has two sockets, coincides with the range of number of elements at which HSW performance drops from just under an efficiency of 13 to approximately 6. If 40 tracers are included, approximately 20 elements/socket fit in L3, which again coincides with the slight drop in HSW efficiency in the case of tracers.

In contrast, KNL and GPU show a strong dependence on workload, evident in both Figs. 7 and 8. For GPU, enough work must be provided to fill all the cores, 2560 in the case of V100, and efficiency is not saturated until each core has multiple parallel loop iterates of work to compute. Nonetheless, the V100 is still reasonably fast even with low workload; for example, at 150 elements and with no tracers, 1 V100 is faster than 1 32-core HSW node. Yet with 150 elements and 16 cores per GPU thread block, only 2400 of the 2560 V100 cores are used, and each for only one loop iterate. However, the Intel Xeon Skylake is substantially faster than any other architecture at this workload. We conclude that high workload per GPU is necessary to exploit the GPU's efficiency.

4.5 GPU kernel performance

In preparation for running on Summit, we compare performance of the code on GPU and CPU. We use nvprof to determine which important kernels are performing well on GPU, and which need improvement. From Figs. 6 and 7, HOMMEXX runs end to end on a V100 device from $1.2\times$ to $3.8\times$ faster than on a HSW node, depending on number of elements, for $n_q = 40$ tracers.

Table 1 breaks down performance by the major algorithm sets: tracer advection (RK Tracers), dynamics (RK Dynamics), hyperviscosity for dynamics (Hypervis. Dyn.), and vertical remap (Vert. Remap). Within each set, it provides data for up to the three longest-running kernels, listed in descending order of runtime. Data were collected for $n_q = 40$; one V100 device and one HSW node with 32 ranks and 2 threads/rank; and $n_e = 8$ and 16, or 384 and 1,536 elements, respectively. Columns are as follows:

$$p \equiv \frac{\text{Alg. GPU Time}}{\text{HOMMEXX Total GPU Time}}; \quad g \equiv \frac{\text{Alg. CPU Time}}{\text{Alg. GPU Time}}.$$

Warp eff. is the ratio of average active threads per warp, the `nvprof` Warp Execution Efficiency metric; and *Occ.* is the percentage of average active warps on a streaming multiprocessor, the Achieved Occupancy metric.

While most of the kernels within an algorithm set are split up by MPI communication rounds, several of the Vertical Remap and one RK Tracer kernel were split up for profiling. Warp efficiency is high in kernels that have computations with few or no conditional statements. RK Tracers Kernel 4 and the Vert. Remap kernels have many conditional statements. Hypervis. Dyn. Kernel 1 has a section focused on vertical levels near the top boundary; this section leaves some threads idle. The RK Dynamics Residual kernel and the Hyperviscosity kernels have low occupancy because of register usage; these are targets for future optimization effort that will focus on increasing the occupancy to 50% through register usage reduction.

5 Conclusions

In this work we presented results of our effort to rewrite an existing Fortran-based code for global atmospheric dynamics to a performance portable version in C++. HOMME is a critical part of E3SM, a globally coupled climate model funded by the

Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-218
Manuscript under review for journal Geosci. Model Dev.
Discussion started: 26 October 2018

Geoscientific Model Development Discussions
Open Access
EGU

Table 1. V100 GPU Performance Study

	384 elems.				1536 elems.	
Kernel	p	g	Warp Eff. (%)	Occ. (%)	p	g
Homme Total	1.0	3.7	——	——	1.0	4.2
RK Tracers	0.69	4.5	——	——	0.73	4.5
Kernel 1	0.13	2.3	92	62	0.14	2.4
Kernel 2	0.093	5.8	99	55	0.1	5.3
Kernel 3	0.065	2.1	91	62	0.07	2.1
RK Dynamics	0.13	1.8	——	——	0.11	3.7
Residual	0.085	1.3	92	47	0.077	2.0
Hypervis. Dyn.	0.11	1.1	——	——	0.091	3.9
Kernel 1	0.021	1.6	91	23	0.019	3.1
Kernel 2	0.022	1.5	91	33	0.021	2.1
Kernel 3	0.018	0.83	53	24	0.015	2.8
Vert. Remap	0.053	4.3	——	——	0.055	4.6
PPM	0.018	4.0	64	46	0.019	3.9
Remap	0.02	1.9	62	24	0.021	2.0
Cell Means	0.0094	6.0	86	46	0.01	6.4

DOE, and this work was part of the effort to prepare E3SM for future exascale computing resources. Our results show that it is indeed possible to write performance portable code using C++ and Kokkos that can match or exceed the performance of a highly-optimized Fortran code on CPU and KNL architectures while also achieving good performance on the GPU.

We presented performance results of the new, end-to-end implementation in HOMMEXX over a range of simulation regimes, and, where possible, compared them against the original Fortran code. Specifically, in a node-to-node comparison, our results show that HOMMEXX is faster than HOMME on dual-socket HSW by up to ∼1.1× and faster on KNL by up to ∼1.3×. These two results establish that HOMMEXX has better end-to-end performance than a highly tuned, highly used code. Next, in a node-to-device comparison, HOMMEXX is faster on KNL than dual-socket HSW by up to ∼2.4×; and HOMMEXX is faster on GPU than on dual-socket HSW by up to ∼3.2×. These two sets of results establish that HOMMEXX has high performance on nonconventional architectures, at parity with or in some cases better than the end-to-end GPU performance reported for other codes for similar applications (Demeshko et al., 2018; Howard et al., 2017).

By leveraging the Kokkos programming model, HOMMEXX is able to expose a large amount of parallelism, which is necessary (though not sufficient) for performance on all target architectures. We also provided some details of our implementation,

and motivated the choices we made on the base of portability and performance. In particular, we highlighted a few important design choices that benefited performance on different architectures, such as the change in data layout, thread count-aware sizing of temporary buffers, and the use of explicit vectorization,

We point out that due to architecture differences, in order to achieve portable performance, a substantial effort was needed, including a careful study of the most computationally loaded routines on each architecture, and the adoption of advanced optimization strategies.

The work here is a significant data point to inform strategic decisions on how to prepare E3SM, and other large scientific research code projects, for new HPC architectures. The results are definitive that the approach we took using C++ and Kokkos does lead to performance portable code ready to run on new architectures. We view the results of reaching and even exceeding parity of HOMMEXX compared to HOMME on CPU and KNL architectures to be a success, based on our initial belief that switching to C++ and having the same code base run well on GPUs was going to hurt performance. (In our proposal, our metric of success was to achieve performance portability and to not be more than 15% slower.) Furthermore, the results regarding the relative performance of the code across numerous architectures as a function of workload have already been instrumental in planning for scientific runs and computer time allocation requests, since different scientific inquiries favor maximum strong-scaled throughput and others favor efficiency.

Decisions on whether to write new components, or to pursue targeted rewrites of existing components, in C++ using Kokkos would need to incorporate many other factors beyond the scope of this paper. The objective results here indicate that it is possible to achieve performance in C++ across numerous architectures. It is anticipated, but not yet shown, that a GPU port of the Fortran HOMME code using OpenACC will also be a path to performance on that architecture. In our discussions on the future programming model for E3SM components, other subjective factors come into play: the availability of code developers and their expertise and preferences, the relative readability and testability of the code, the anticipated portability to future architectures, and the like.

Code and data availability. The source code is publicly available at Bertagna et al. (2018). Performance data in form of raw timers is available in the same repository in folder `perf-runs/sc18-results` together with building running scripts.

Author contributions. All authors contributed to the content of the manuscript: All authors contributed to introduction, literature overview and conclusions. L. Bertanga, A. Bradley, M. Deakin, O. Guba, A. Salinger, and D. Sunderland contributed to overview of implementation and performance results. M. Taylor, A. Bradley and O. Guba contributed to the section on HOMME dycore. L. Bertanga, A. Bradley, M. Deakin, O. Guba, and D. Sunderland contributed to code base of HOMMEXX 1.0 and performance runs. Irina K. Tezaur provided infrastructure support and testing. A. Bradley, A. Salinger, M. Taylor, and I. Tezaur provided expertise in high-performance computing, expertise in dycore modeling, and leadership.

Competing interests. The authors declare that they have no conflict of interest.

Acknowledgements. The authors are grateful to Si Hammond, Kyungjoo Kim, Eric Phipps, Steven Plimpton and the Kokkos team for their suggestions. This work was part of the CMDV program, funded by the U.S. Department of Energy (DOE) Office of Biological and Environmental Research. Sandia National Laboratories is a multimission 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. This research used resources of the National Energy Research Scientific Computing Center, a User Facility supported by the Office of Science of DOE under Contract No. DE-AC02-05CH11231.

References

Energy Exascale Earth System Model, https://climatemodeling.science.energy.gov/projects/energy-exascale-earth-system-model.

Technical Specification for C++ Extensions for Parallelism, http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/n4507.pdf.

The OpenACC Application Programming Interface, https://www.openacc.org/sites/default/files/inline-files/OpenACC.2.6.final.pdf, 2017.

Abdi, D., Giraldo, F., Constantinescu, E., Carr, L., Wilcox, L., and Warburton, T.: Acceleration of the IMplicit–EXplicit nonhydrostatic unified model of the atmosphere on manycore processors, The International Journal of High Performance Computing Applications, https://doi.org/10.1177/1094342017732395, https://doi.org/10.1177/1094342017732395, 2017.

Bertagna, L., Deakin, M., Guba, O., Sunderland, D., Bradley, A., Tezaur, I. K., Taylor, M., and Salinger, A.: E3SM-Project/HOMMEXX: Release-v1.0.0, https://doi.org/10.5281/zenodo.1256256, https://doi.org/10.5281/zenodo.1256256, 2018.

Bhanot, G., Dennis, J. M., Edwards, J., Grabowski, W., Gupta, M., Jordan, K., Loft, R. D., Sexton, J., St-Cyr, A., Thomas, S. J., Tufo, H. M., Voran, T., Walkup, R., and Wyszogrodski, A. A.: Early experiences with the 360TF IBM Bue Gene/L platform, International Journal of Computational Methods, 05, 237–253, https://doi.org/10.1142/S0219876208001443, 2008.

Callahan, D., Dongarra, J., and Levine, D.: Vectorizing compilers: a test suite and results, Proceedings of the 1988 ACM/IEEE conference on Supercomputing (Supercomputing '88), pp. 98–105, 1988.

Canuto, C., Hussaini, M., Quarteroni, A., and Zang, T.: Spectral Methods: Evolution to Complex Geometries and Applications to Fluid Dynamics, Scientific Computation, Springer Berlin Heidelberg, 2007.

Carpenter, I., Archibald, R. K., Evans, K. J., Larkin, J., Micikevicius, P., Norman, M., Rosinski, J., Schwarzmeier, J., and Taylor, M. A.: Progress towards accelerating HOMME on hybrid multi-core systems, International Journal of High Performance Computing Applications, 27, 335–347, 2013.

Carter Edwards, H., Trott, C. R., and Sunderland, D.: Kokkos: Enabling manycore performance portability through polymorphic memory access patterns, J. Parallel Distr. Com., 74, 3202–3216, 2014.

Colella, P. and Woodward, P. R.: The Piecewise Parabolic Method (PPM) for gas-dynamical simulations, Journal of Computational Physics, 54, 174 – 201, https://doi.org/https://doi.org/10.1016/0021-9991(84)90143-8, http://www.sciencedirect.com/science/article/pii/0021999184901438, 1984.

Demeshko, I., Watkins, J., Tezaur, I. K., Guba, O., Spotz, W. F., Salinger, A. G., Pawlowski, R. P., and Heroux, M. A.: Toward performance portability of the Albany finite element analysis code using the Kokkos library, The International Journal of High Performance Computing Applications, https://doi.org/10.1177/1094342017749957, https://doi.org/10.1177/1094342017749957, 2018.

Dennis, J., Fournier, A., Spotz, W. F., St-Cyr, A., Taylor, M. A., Thomas, S. J., and Tufo, H.: High-Resolution Mesh Convergence Properties and Parallel Efficiency of a Spectral Element Atmospheric Dynamical Core, The International Journal of High Performance Computing Applications, 19, 225–235, https://doi.org/10.1177/1094342005056108, 2005.

Dennis, J., Edwards, J., Evans, K. J., Guba, O., Lauritzen, P. H., Mirin, A. A., St-Cyr, A., Taylor, M. A., and Worley, P.: CAM-SE: A scalable spectral element dynamical core for the Community Atmosphere Model, Journal of High Performance Computing Applications, 26, 74–89, 2012.

Estérie, P., Gaunard, M., Falcou, J., Lapresté, J.-T., and Rozoy, B.: Boost.SIMD: Generic programming for portable SIMDization, 2012 21st International Conference on Parallel Architectures and Compilation Techniques (PACT), pp. 431–432, 2012.

Fu, H., Liao, J., Xue, W., Wang, L., Chen, D., Gu, L., Xu, J., Ding, N., Wang, X., He, C., Xu, S., Liang, Y., Fang, J., Xu, Y., Zheng, W., Xu, J., Zheng, Z., Wei, W., Ji, X., Zhang, H., Chen, B., Li, K., Huang, X., Chen, W., and Yang, G.: Refactoring and Optimizing

the Community Atmosphere Model (CAM) on the Sunway TaihuLight Supercomputer, in: SC16: International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 969–980, https://doi.org/10.1109/SC.2016.82, 2016.

Fu, H., Liao, J., Ding, N., Duan, X., Gan, L., Liang, Y., Wang, X., Yang, J., Zheng, Y., Liu, W., Wang, L., and Yang, G.: 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, SC '17, pp. 1:1–1:12, ACM, New York, NY, USA, https://doi.org/10.1145/3126908.3126909, http://doi.acm.org/10.1145/3126908.3126909, 2017.

Fuhrer, O., Osuna, C., Lapillonne, X., Gysi, T., Cumming, B., Bianco, M., Arteaga, A., and Schulthess, T.: Towards a performance portable, architecture agnostic implementation strategy for weather and climate models, Supercomputing Frontiers and Innovations, 1, http://superfri.org/superfri/article/view/17, 2014.

Fuhrer, O., Chadha, T., Hoefler, T., Kwasniewski, G., Lapillonne, X., Leutwyler, D., Lüthi, D., Osuna, C., Schär, C., Schulthess, T. C., and Vogt, H.: Near-global climate simulation at 1 km resolution: establishing a performance baseline on 4888 GPUs with COSMO 5.0, Geoscientific Model Development Discussions, 2017, 1–27, https://doi.org/10.5194/gmd-2017-230, https://www.geosci-model-dev-discuss.net/gmd-2017-230/, 2017.

Grant, R. E., Laros III, J. H., Levenhagen, M., Olivier, S. L., Pedretti, K., Ward, L., and Younge, A. J.: FY17 CSSE L2 Milestone Report: Analyzing Power Usage Characteristics of Workloads Running on Trinity, 2017.

Guba, O., Taylor, M., and St-Cyr, A.: Optimization-based limiters for the spectral element method, Journal of Computational Physics, 267, 176 – 195, https://doi.org/https://doi.org/10.1016/j.jcp.2014.02.029, 2014a.

Guba, O., Taylor, M. A., Ullrich, P. A., Overfelt, J. R., and Levy, M. N.: The spectral element method on variable resolution grids: evaluating grid sensitivity and resolution-aware numerical viscosity, Geoscientific Model Development Discussions, 7, 4081–4117, https://doi.org/10.5194/gmdd-7-4081-2014, http://www.geosci-model-dev-discuss.net/7/4081/2014/, 2014b.

Harris, M.: Developing Portable CUDA C/C++ Code with Hemi, http://devblogs.nvidia.com/parallelforall/developing-portable-cuda-cc-code-hemi/, 2015.

Hornung, R. D. and Keasler, J. A.: The RAJA Portability Layer: Overview and Status, Tech. rep., Lawrence Livermore National Laboratory (LLNL), Livermore, CA, 2014.

Howard, M., Bradley, A., Bova, S. W., Overfelt, J., Wagnild, R., Dinzl, D., Hoemmen, M., and Klinvex, A.: Towards performance portability in a compressible cfd code, in: 23rd AIAA Computational Fluid Dynamics Conference, p. 4407, 2017.

Hurrell, J. W., Holland, M. M., Gent, P. R., Ghan, S., Kay, J. E., Kushner, P. J., Lamarque, J, F., Large, W. G., Lawrence, D., Lindsay, K., Lipscomb, W. H., Long, M. C., Mahowald, N., Marsh, D. R., Neale, R. B., Rasch, P., Vavrus, S., Vertenstein, M., Bader, D., Collins, W. D., Hack, J. J., Kiehl, J., and Marshall, S.: The community earth system model: A framework for collaborative research, Bulletin of the American Meteorological Society, 94, 1339–1360, https://doi.org/10.1175/BAMS-D-12-00121.1, 2013.

Intel(R): Fortran Array Data and Arguments and Vectorization, https://software.intel.com/en-us/articles/fortran-array-data-and-arguments-and-vectorization, a.

Intel(R): Intel Intrinsics Guide, https://software.intel.com/sites/landingpage/IntrinsicsGuide, b.

Kaiser, H., Brodowicz, M., and Sterling, T.: ParalleX An Advanced Parallel Execution Model for Scaling-Impaired Applications, in: Proceedings of the 2009 International Conference on Parallel Processing Workshops, 2009.

Kale, L. V., Bohm, E., Mendes, C. L., Wilmarth, T., and Zheng, G.: Programming Petascale Applications with Charm++ and AMPI, in: Petascale Computing: Algorithms and Applications, 2008.

Maday, Y. and Patera, A. T.: Spectral element methods for the incompressible Navier-Stokes equations, in: State-of-the-art surveys on computational mechanics (A90-47176 21-64). New York, American Society of Mechanical Engineers, 1989, p. 71-143. Research supported by DARPA., pp. 71–143, 1989.

Medina, D., St-Cyr, A., and Warburton, T.: OCCA: A unified approach to multi-threading languages, SIAM Journal on Scientific Computing (SISC), 2014.

Norman, M., Larkin, J., Vose, A., and Evans, K.: A case study of CUDA FORTRAN and OpenACC for an atmospheric climate kernel, Journal of Computational Science, 9, 1–6, https://doi.org/https://doi.org/10.1016/j.jocs.2015.04.022, http://www.sciencedirect.com/science/article/pii/S1877750315000605, computational Science at the Gates of Nature, 2015.

NVIDIA(C): NVIDIA Tesla P100, https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf.

Rajan, M., Doerfler, D., Tupek, M., and Hammond, S.: An Investigation of Compiler Vectorization on Current and Next-generation Intel Processors using Benchmarks and Sandia's SIERRA Applications, https://escholarship.org/uc/item/05x9132w, 2015.

Salinger, A., Bartlett, R., Bradley, A., Chen, Q., Demeshko, I., Gao, X., Hansen, G., Mota, A., Muller, R., Nielsen, E., Ostien, J., Pawlowski, R., Perego, M., Phipps, E., Sun, W., and Tezaur, I.: Albany: Using Agile Components to Develop a Flexible, Generic Multiphysics Analysis Code, Int. J. Multiscale Comput. Engng., 14, 415–438, 2016.

Spotz, W. F., Smith, T. M., Demeshko, I. P., and Fike, J. A.: Aeras: A Next Generation Global Atmosphere Model, Procedia Computer Science, 51, 2097 – 2106, https://doi.org/https://doi.org/10.1016/j.procs.2015.05.478, http://www.sciencedirect.com/science/article/pii/S1877050915012867, international Conference On Computational Science, ICCS 2015, 2015.

Taylor, M.: Conservation of Mass and Energy for the Moist Atmospheric Primitive Equations on Unstructured Grids, in: Numerical Techniques for Global Atmospheric Models, edited by Lauritzen, P. H., Jablonowski, C., Taylor, M. A., and Nair, R. D., Springer, 2012.

Worley, P. H., Craig, A. P., Dennis, J. M., Mirin, A. A., Taylor, M. A., and Vertenstein, M.: Performance and Performance Engineering of the Community Earth System Model, in: Proceedings of the ACM / IEEE Supercomputing SC'2011 Conference, 2011.