Final Report for the NPS-NRL-Rice-UIUC Collaboration on Navy Atmosphere-Ocean Coupled Models on Many-Core Computer Architectures

Lucas C. Wilcox
Department of Applied Mathematics
Naval Postgraduate School
833 Dyer Road; Monterey, CA 93943 USA
phone: (831) 656-3249 fax: (831) 656-2355 e-mail: lwilcox@nps.edu

Francis X. Giraldo
Department of Applied Mathematics
Naval Postgraduate School
833 Dyer Road; Monterey, CA 93943 USA
phone: (831) 656-2293 fax: (831) 656-2355 e-mail: fxgirald@nps.edu

Timothy Campbell
Naval Research Laboratory
Oceanography Division, Code 7322; Stennis Space Center, MS 39529 USA
phone: (228) 688-5284 e-mail: tim.campbell@nrlssc.navy.mil

Andreas Klöckner
Department of Computer Science
University of Illinois at Urbana-Champaign
201 North Goodwin Avenue; Urbana, IL 61801 USA
phone: (217) 244-6401 e-mail: andreask@illinois.edu

Timothy Warburton
Mathematics Department
Virginia Tech
225 Stanger Street; Blacksburg, VA 24061-0123 USA
phone: (540) 231-5960 e-mail: tcew@math.vt.edu

Timothy Whitcomb
Naval Research Laboratory
7 Grace Hopper Ave, MS2; Monterey, CA 93943 USA
phone: (831) 656-4812 fax: (831) 656-4769 email: tim.whitcomb@nrlmry.navy.mil

Award Number: N00014-15-WX-00694; N00014-15-WX-01603; N00014-16-WX-01504; N00014-16-1-2488; N00014-14-1-0117; N00014-14-AF-00002

EXECUTIVE SUMMARY

In this three-year project (2014-2016) the Principal Investigators (PI) were able to achieve 7 out of the 10 original objectives of the proposal. In addition, we added an extra objective in the third (last) year of the project. Let us now describe the objectives of the project and explain why some were not met and for those that were, what the significance of these accomplishments mean.

(1) The first objective of the project was to identify the bottlenecks in the NUMA model. We identified the bottlenecks in NUMA the first year of the project, which allowed us to form a roadmap for how to tackle the work for the rest of the project. (2) The second objective was to port the explicit dynamics of NUMA to many-core architectures. Here, by explicit dynamics we mean the spatial discetization methods (the continuous and discontinuous Galerkin, CG/DG, flow solvers for the fully compressible Navier-Stokes equations) without worrying about semi-implicit time-integrations and all the machinery that this requires. By many-core computers we mean, e.g., Nvidia GPUs and Intel's Xeon Phis (Knights Landing). This second objective was achieved within the first two years of the project. We were able to achieve this objective thanks to our *hardware-agnostic* approach using the Loo.py and OCCA libraries which were improved in the course of this project. The idea behind our *hardware-agnostic* approach is that we only want to write the compute-kernels for a given model just once using a specialized language (in this case OCCA with Loo.py for optimization). Then, when we wish to run our model on a new hardware, all we need to do is to write a new backend that will translate the OCCA kernel to a hardware-specific language (e.g., CUDA on Nvidia hardware and OpenMP on Intel hardware). (3) The third objective was to extend the explicit dynamics on many-core to vertically implicit dynamics (e.g., introducing semi-implicit time-integrators into the vertical direction as is typically done in mesoscale models). We achieved this goal in years 2 and 3 of the project for both the CG and DG methods for various vertically implicit-explicit methods (such as implicit multi-step and multi-stage methods). (4) Although not in the original proposal, we were able to extend the vertically implicit time-integration on many-core to implicit time-integration in all directions. The reason why this was not originally proposed is because this approach was not expected to be competitive with the vertically implicit only approach. However, as we pushed NUMA to very fine grid resolution (we ran 3 km global resolutions in this work) the advantages of vertically implicit-only disappear and begin to favor the implicit in all directions approach. To our knowledge, this is the first time that the full dynamics (including implicit time-integration) of an atmospheric model has been ported to many-core architectures (which includes GPUs and Intel's Xeon Phis). The reason why few groups attempt this is because this requires porting not only the (i) spatial discretization methods, (ii) time-integrators, but also the (iii) iterative solvers, and (iv) preconditioners. In sum, there is an inordinate amount of machinery and infrastructure that has to be ported to many-core computers. Again, we were able to achieve this feat by taking advantage of the *hardware-agnostic* approach offered by the OCCA library. (5) The fifth objective of the original proposal was to port relevant physics onto the GPU. However, at the first PI meeting we learned that one of the funded groups was porting the RRTMG radiation code to the GPU. Furthermore, the NEPTUNE group at NRL-Monterey was coordinating a GPU-effort for the NEPTUNE [1] physics with the University of Utah. For these reasons, it was decided to drop this work from the proposal. (6) The sixth objective was to transition all the compute-kernels

[1] NEPTUNE is the Naval Research Laboratory's next-generation weather model that uses the NUMA dynamics as its flow solver.

developed in this project into the NUMA model which was achieved throughout the course of the project. The main trunk of NUMA (which sits in a GIT repository at NPS) contains both a CPU-only version and a many-core version, all within the same code-base. (7)-(8) Although Loo.py already existed before this project began, we adapted Loo.py as needed in order to make the OCCA kernels run faster on various types of hardware. By the end of the project, the Loo.py team was able to show that Loo.py could generate compute-kernels (from basic OCCA kernels) that were as fast (or faster) than the best "hand-written" OCCA kernels. One of the objectives of this part of the work was to show that semi-auto-tuned code-generation is possible. We call it "semi-auto-tuned" because Loo.py does not work in isolation but, rather, requires intervention from the code-developer to instruct it on which aspects of the code to optimize (e.g., loop unrolling, data layout, etc.). (9) In the original proposal, we proposed to implement ESMF interfaces to many-core architectures. However, at the first PI meeting we learned that another group was funded to carry out this work, which led us to drop this work from our project. (10) This item was originally proposed as implementing the NUMA model as an ESMF component. However, after the first PI meeting, the NRL scientists in our proposal agreed that it was best to replace NUMA with NEPTUNE for this portion of the work. This part of the workflow was entirely independent from the many-core work being tackled by NPS, UIUC, and Rice Universities. Some progress was made on this front but it was not funded by this grant. (11) The last item proposed in our project was to assess the performance of our many-core approach. To this end, PI Giraldo and one postdoc attended a one-week OpenACC "hackathon" sponsored by Oak Ridge National Laboratory where Giraldo and his team ported the explicit dynamics of NUMA to the Titan supercomputer using OpenACC and compared it to the OCCA kernels. The OCCA kernels were much faster and for this reason we decided that OpenACC would not work as well for the NUMA model. We have also completed three journal publications on the performance of NUMA on various computer architectures including: (i) porting NUMA onto the Mira supercomputer (the sixth fastest computer in the world) showing that NUMA achieves perfect strong scaling up to the full machine (over 3 million MPI ranks); (ii) porting NUMA to the Titan supercomputer (the 3rd fastest computer in the world) using almost 90% of the Nvidia GPUs that Titan has to offer and showing that NUMA with OCCA achieves 90% weak scaling for both explicit, vertically implicit, and implicit in all directions. The simulations on Titan were achieved using the OCCA kernels with CUDA backends. In addition, NUMA was also shown to achieve 90% weak scaling on the Intel Xeon Phis (Knights Landing), also using the OCCA kernels with OpenMP backends.

One of our goals for this proposal was to help facilitate the creation of a working group that would collaborate or, at the very least, discuss the vices and virtues of various approaches for porting existing Earth System Models to many-core architectures. PI Giraldo helped co-author a position paper from the NUOPC (ESPC) Common Model Architecture (CMA) group for presentation to the National Science Foundation working group on the needs of the Earth Systems Modeling community for getting these codes ready for the exascale through the National Strategic Computing Initiative plan. In fact, many of the lessons learned in our project went into the supporting briefing slides of that report.

In the report that follows, we focus on the highlights of the work achieved during the three years of the project.

LONG-TERM GOALS

The ever increasing pace of improvement in the state-of-the-art high performance computing technology promises enhanced capabilities for the next-generation atmospheric models. In this project we primarily consider incorporating state of the art numerical methods and algorithms to enable the Nonhydrostatic Unified Model of the Atmosphere (http://frankgiraldo.wix.com/mysite/numa), also known as NUMA, to fully exploit the current and future generations of parallel many-core computers. This includes sharing the tools developed for NUMA (through open-source) with the U.S. community that can synergistically move the knowledge of accelerator-based computing to many of the climate, weather, and ocean modeling laboratories around the country.

WORK COMPLETED

In the course of this project we completed the following items:

1. identified bottlenecks in the NUMA model;
2. ported the explicit time-integration portion of the dynamics onto many-core devices;
3. ported the implicit-in-the-vertical dynamics onto many-core devices;
4. ported fully implicit-explicit time-integration (in all three-dimensions) dynamics onto many-core devices - the challenge here was to port all of the iterative solvers, preconditioners, and the fully three-dimensional implicit machinery (highlighted in red in Table 1);
5. transition many-core kernels into NUMA;
6. adapt the Loo.py code generator for the needs of the project;
7. develop a source-to-source translation capability built on Loo.py to facilitate the NUMA transition;
8. assess performance against current modeling suite.

The management plan for these work items is shown in Table 1. Below is a summary of the work completed for these items.

Identifying bottlenecks in NUMA In the first two years of the project, we completed two major tasks in identifying bottlenecks in NUMA. The first is the incorporation of `p4est`[2] into NUMA to enable efficient mesh generation on large distributive memory machines. The second is the unification of the CG and DG NUMA codes in preparation for using the kernels developed in the mini-apps.

As NRL Monterey was using NUMA, it became clear that mesh generation was a bottleneck for scaling NUMA on large distributive memory systems. During this collaboration, we (along with

[2] `p4est` is a C library to manage a collection (a forest) of multiple connected adaptive quadtrees or octrees in parallel. It can be used to efficiently generate adapted computational meshs needed for CG and DG codes like NUMA. The code is open-source and more information can be found at http://p4est.org.

Activity	Months:	0–12	13–24	25–36
Identifying bottlenecks		✓		
Explicit dynamics		✓	✓	
Vertically-implicit dynamics			✓	✓
3D Implicit-Explicit dynamics				✓
Transition many-core kernels into NUMA			✓	✓
Adapt Loo.py		✓	✓	✓
Develop source translation tool			✓	✓
Many-core CPU Kernels				✓
Assess Performance		✓	✓	✓
Disseminate work (Publications)			✓	✓

Table 1: Time-line of proposed activities for different months of the project.

Dr. Tobin Isaac and other members of the NUMA team) incorporated `p4est` into NUMA (for more details see [9]).

The current version of NUMA in NEPTUNE only has continuous Galerkin (CG) discretizations and thus only a particular CG data layout. A separate, scalable, discontinuous Galerkin version of NUMA has been folded into the master branch of NUMA. This new version of NUMA is capable of handling two data layouts for CG and one for DG. This unification of all NUMA codes into a single code-base will greatly simplify the further improvement of the model (for more details about the unified NUMA code see [3]).

Explicit Dynamics The original CG/DG mini-app, `gNUMA`, with the hand-written OCCA kernels has been fully incorporated and extended in NUMA (for more details see [5]).

Based on the scalability success of incorporating `p4est` into NUMA and looking to the future of supporting adaptive-mesh refinement (AMR), we decided to use `p4est` in our mini-app. Doing this, we have created a new mini-app, `bigNUMA` (`bfam`[3] integrated `gNUMA`), which currently solves the Euler equations (with local DG based diffusion) using a DG discretization on multiple accelerator devices with MPI+OCCA.

The host code of `bigNUMA` differs from `gNUMA` but they currently share the same device code. Working with NPS Assistant Professor Jeremy Kozdon (one of the authors of `bfam`) we have developed the infrastructure needed to build *hp*-adaptive DG discretizations using OCCA on multiple devices using MPI which includes dynamic grid adaptivity. The bigNUMA mini-app has been extended and incorporated into NUMA by Giraldo and Kozdon under a different ONR grant.

Implicit-Explicit (IMEX) Time-Integration of the Dynamics Operational numerical weather prediction software often use IMplicit-EXplicit (IMEX) methods where the process in the vertical direction is solved implicitly while that in the horizontal direction is solved explicitly.

[3] `bfam` is a set of tools to develop coupled discontinuous Galerkin and multi-block summation-by-parts methods to solve multi-physics problems which uses `p4est` for mesh manipulation. More information can be found at http://bfam.in/

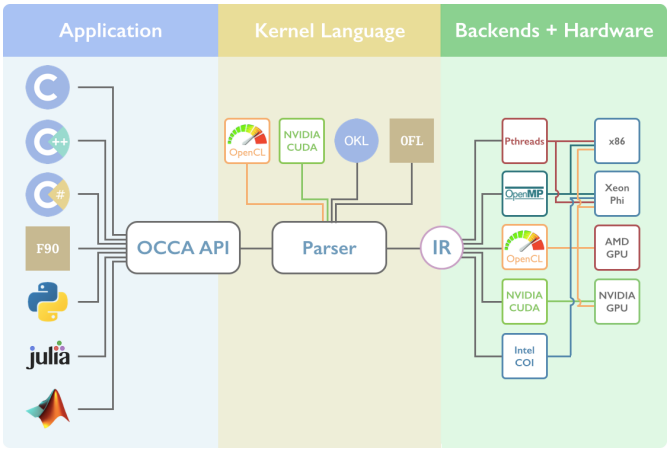


Figure 1: Overview of OCCA API, kernel languages, and programming model support. See libocca.org for tutorials and documentation

.

IMEX time integration methods sidestep the constraint imposed by the Courant-Friedrichs-Lewy condition on explicit methods through corrective implicit solves within each time step. Abdi and Giraldo fully ported all the IMEX methods available in NUMA to many-core processors. The unconditional stability (relative to the speed of sound) of IMEX means the relative speedup of IMEX over explicit methods can increase substantially with the Courant number as long as the accuracy of the solution is acceptable; for example, for some simulations it is possible to achieve a speedup of 100X over explicit methods when using Courant numbers of, say, C=150 [4]. Several improvements on the IMEX procedure were necessary in order to improve the performance obtained with explicit methods such as: a) reducing the number of degrees of freedom of the IMEX formulation by forming the Schur complement; b) formulating a horizontally-explicit vertically-implicit (HEVI) 1D-IMEX scheme that has a lower workload and potentially better scalability than 3D-IMEX; c) using high-order polynomial preconditioners to reduce the condition number of the resulting system of the fully 3D-IMEX problem; d) using a direct solver for the 1D-IMEX method which has a constant cost for any Courant number. Without *all* of these improvements, explicit time integration methods turned out to be difficult to beat. We have validated our results with standard benchmark problems in numerical weather prediction and evaluated the performance and strong scalability of the IMEX method using up to 4192 GPUs. The results regarding the implementation of the IMEX methods in NUMA on many-core architectures can be found in [4].

OCCA Over the course of this project various contributions and enhancements have been made to OCCA focused on improving the robustness, capabilities, and usability of the OCCA many-core portable parallel threading run-time library. The OCCA library contributes a component to the efforts of the ESPC-AOLI NOPP team to "future proof" simulation frameworks against upheavals in many-core programming models and many-core architectures.

OCCA offers multiple language API support for programmers. Library support exists for C, C++,

[4]The Courant number given is with respect to the fastest waves in the system - for the Euler equations it represents the Courant number based on the acoustic wave speed.

C#, Python, Julia, and MATLAB. These interfaces have been complemented with a native F90 frontend to accommodate the atmospheric modeling community preference for Fortran. See Figure 1 for an overview of the OCCA infrastructure. Documentation of the OCCA system is underway at http://libocca.org. OCCA adopts the accelerator offload model of OpenCL and CUDA but extends this approach to allow a single kernel to be executed through OpenMP, CUDA, OpenCL, and pThreads.

To allow for using many-core CPUs we included a new backend using the Intel SPMD Program Compiler (ISPC). ISPC is a compiler that supports single instruction, multiple data programming vectorization on Intel CPUs and the Xeon Phi. We have implemented a basic ISPC OCCA backend that we are using to explore the performance of the gNUMA mini-app kernels on CPUs.

We also explored using the OpenMP backend for use on the Knights Landing (KNL) with which we obtained good results that are comparable to our previous results using the GPU. The existing GPU kernels had to be modified to work with OpenMP and a different approach of using one-element-per thread instead of the one-node-per-thread approach of the GPU. Moreover, it was necessary to have a fully vectorized code (16 floats/ 8 doubles can be processed in one go), when using the OpenMP approach on the KNL. We have re-written our kernels to work with float1, float4 and float8 vector intrinsics and obtained speedups of upto 2.5X.

Transition many-core kernels into NUMA This development initially began with kernels from the min-apps `gNUMA`. However, many of the kernels had to be rewritten in order to make them better integrated into the NUMA code (e.g., the volume kernels). In addition, new kernels for the IMEX solvers had to be written from scratch and are now part of the full NUMA software that is ready for distribution to the U.S. Navy (when the need arises).

After year 2, the first stage of unification of the CG and DG codes was completed [3]. This resulted in a single code base that can solve the Euler equations using either CG or DG. Using this as a base the development of an OCCA enabled version of the explicit code has been completed [5]. The extension to vertically-implicit (1D-IMEX) and implicit in both the horizontal and vertical (3D-IMEX) were also completed and presented in [4].

Adapt Loo.py and develop source translation tool The objective for this part of the work was to provide a path towards performance-portable code that is easier and more approachable than writing GPU code by hand, yet provides more control and better performance than is possible with fully automatic approaches [7]. A newly found perspective is that Fortran will be used as a sort of domain-specific language, precisely specifying what, mathematically speaking, is to be computed, while at the same time algorithmic and performance aspects are specified in separate, transformation-based code. Transformation machinery has been built that can exactly match the loop and parallelization structure of a highly optimized hand-written kernel [6]. We have now reproduced the handwritten GPU derivative kernels from gNUMA in Loo.py effectively splitting the algorithm for the schedule of instructions [8].

Many-core CPU Kernels To satisy the need for porting Earth System Models onto many-core CPU and Xeon Phi compute-kernels we explored possible kernel backends. In a perfect

world OpenCL would have been such a backend but it is unclear if Intel will continue supporting this going forward on the Xeon Phi. So we move to an ISPC backend that supports Xeon and Xeon Phi many-core devices. We further added support to Loo.py to output ISPC kernels which are executed with the OCCA host API.

We started by implementing micro-benchmarks, such as Sustainable Memory Bandwidth in High Performance Computers (STREAM), to understand the different performance characteristics of kernels running on the these devices instead of GPUs. As our kernels are memory bound, it quickly became apparent that non-uniform memory access was going to be one of the main issues that will hinder our kernels. We decided to run our kernels with one MPI process per non-uniform memory access domain to avoid these non-uniform memory access issues.

We used Loo.py's transformations to explore the performance of `gNUMA` kernels on many-core CPUs such as the Xeon Phi. For the differentiation kernel we investigated, we were within a factor of two of using the maximum memory bandwidth.

In addition, our approach of using OpenMP + Vectorization (float4/float8) directly within the NUMA code-base has given us very good results on many-core CPU and the Intel Xeon Phi (Knights Landing). Below, we discuss the roofline plots and vectorization results of this portion of this work. The approach we use for vectorization is unique in that we vectorize across the tracers. NUMA often has 8 or more variables to solve, so operations such as computing derivatives, divergence, gradient and curl operators can be done for all of the 8 variables at once. We tested this approach against standard loop-vectorization and found it to give much better results.

Assess Performance Throughout the course of this project, we continuously assessed the performance of all of the different strategies for running code on many-core processors. This assessment took place in the scientific literature through the publication of the MPI+CUDA and MPI+OpenMP works in [5].

Communications with the community The communication with the community has been mainly done through publications and conference attendance. For example, Dr. Daniel Abdi (a key member of PI Giraldo's research group) presented the results of the NUMA many-core work at the NCAR Multi-Core Workshop in September 2016 and at the American Geophysical Union Fall Meeting in December 2016. In addition, PI Giraldo contributed to a NUOPC (ESPC) Common Modeling Architecture position paper on the needs of the Earth Systems Modeling community for presentation to the National Science Foundation. The position paper was crafted by representatives from Navy (PI Giraldo), NOAA, and NASA, to name a few organizations. Many of the lessons learned which were highlighted in the accompanying briefing slides originated from the work on porting NUMA to many-core computers funded by this project.

RESULTS

Here we list some of the main results since the beginning of the project.

NUMA on CPU-based Computers

NUMA has been given the capability of using `p4est` as a mesh/communication pattern generator which has enabled NUMA to scale to over 777,600 cores using 3.1 million MPI threads on Mira, Argonne's 10-petaflops IBM Blue Gene/Q system [9]. In addition, this capability was transitioned to NRL Monterey and was a critical component to NEPTUNE's number one ranking for the strong scalability benchmark in the "technical evaluation of HPC suitability and readiness of five Next Generation Global Prediction System (NGGPS) candidate models to meet operational forecast requirements at the National Weather Service."[5]

NUMA on Nvidia GPUs

To get NUMA ready for many-core devices, we began the transition from CPU-based to many-core based with the mini-apps developed in the first year of the project (e.g., gNUMA). However, as the many-core version of NUMA matured we had to write new OCCA kernels in order to port ALL of the NUMA model in order to fully exploit many-core computers. The first result with NUMA using OCCA (OKL) is presented in [5]. In that paper we benchmarked NUMA in two ways: (1) using a weak scaling study on the Titan supercomputing system at Oak Ridge National Laboratory; and (2) by comparing the single node kernels to a roofline performance model.

Running on GPUs, the multi-device implementation of NUMA with the OCCA kernels reuses the tested and proven method for indirect communication between GPUs by copying data to the CPUs. The scalability of this multi-device implementation has been tested on a GPU cluster, namely, the Titan supercomputer which has 18688 Nvidia Tesla K20 GPU accelerators. The scalability result in Figure 2 shows that NUMA with the OCCA kernels is able to achieve 90% weak scaling efficiency on tens of thousands of GPUs. Different implementations of the unified CG/DG model were tested, where the best performance was achieved by DG with overlapping computation-communication to hide latency. The single node performance of the computation kernels in NUMA has also been evaluated and are presented in Figure 3. More details about the single node and parallel performance can be found in [5].

The numerical weather prediction community is mostly interested in strong-scalability results due to limitations placed on simulation time for a one day forecast (e.g., NOAA's 4.5 minutes wallclock time for a 1 day forecast). Therefore, we evaluate the strong scalability performance of NUMA, first, using explicit time-integration. For this test, we consider three grid resolutions, namely 13 km, 10 km and 3 km, on a cubed sphere grid. The results in Fig. 4 show that 512, 1024 and more than 4096 Nvidia K20X GPUs are required to meet a hypothetical 100 minutes wallclock time limit for a one day forecast. This motivated us to implement IMEX methods in NUMA in order to reduce the time-to-solution by increasing the time step size. This was a significant undertaking as IMEX requires a large amount of infrastructure. PI Giraldo and Dr. Emil Constantinescu (Argonne National Laboratory) implemented the original CPU-based IMEX infrastructure in NUMA, but the challenge in the course of this grant was to port all of that infrastructure to the GPU which was tackled by Dr. Daniel Abdi (a member of PI Giraldo's research group). The results in Fig. 8 show that 3D-IMEX gives a constant speedup of 5X over the RK method and brings down the target wallclock time limit to 20 minutes for the same number of GPUs. In principle, it is possible to reduce the memory consumption of 1D-IMEX, which has given us upto 100X speedup, to work with the problem size and achieve the 4.5 minutes wallclock time

[5] http://www.nws.noaa.gov/ost/nggps/DycoreTestingFiles/AVEC%20Level%201%20Benchmarking%20Report%2008%2020150602.pdf

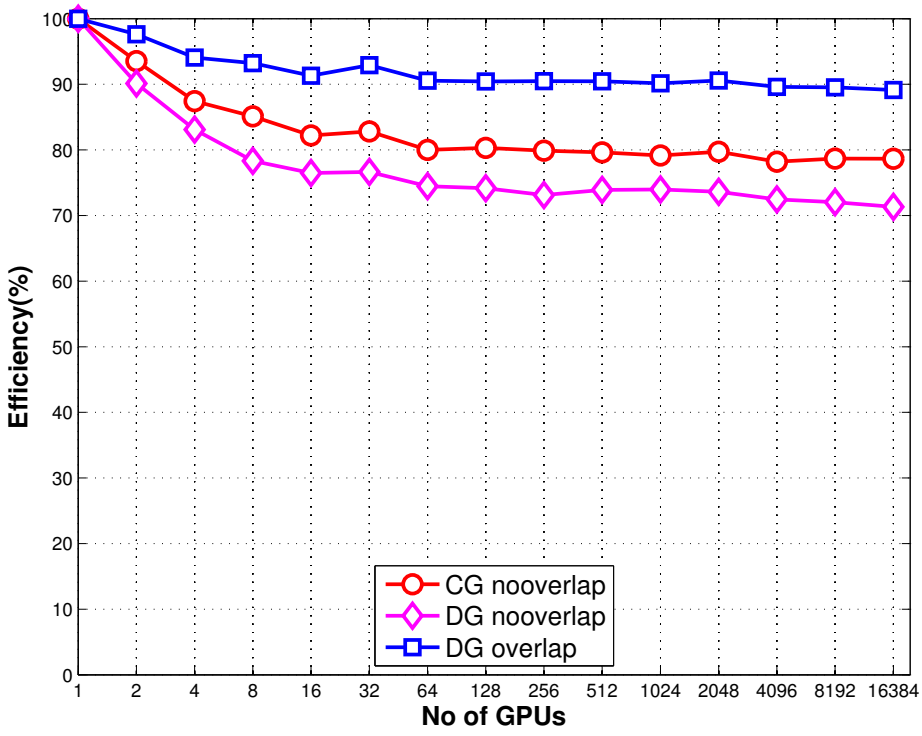
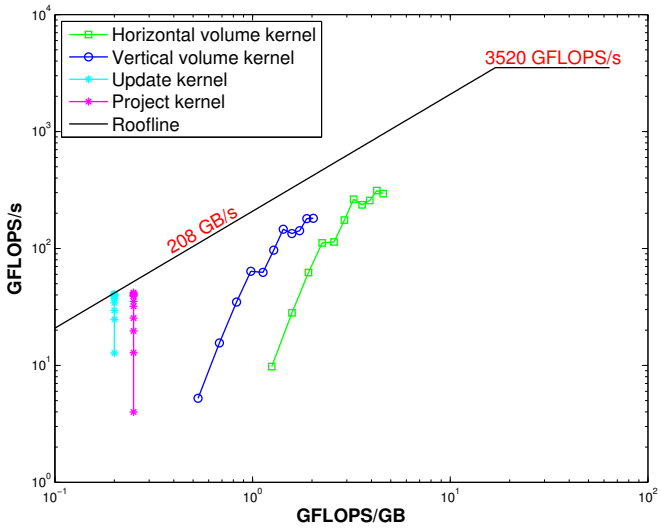
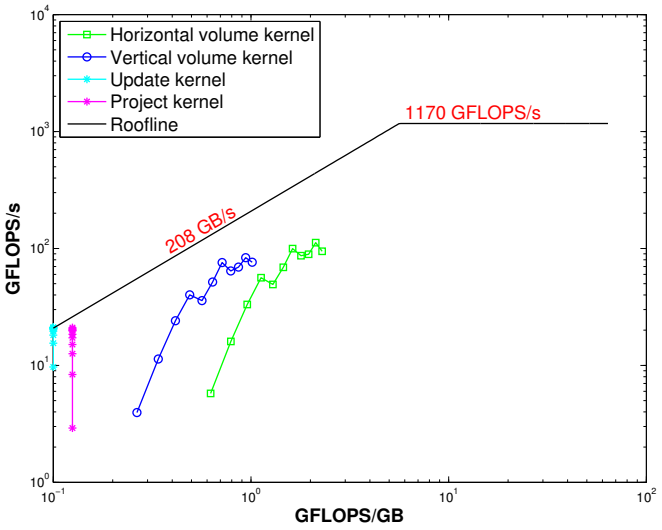


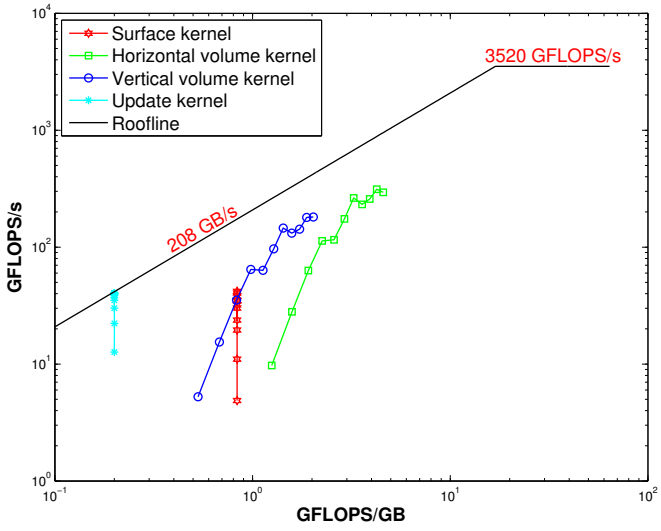
Figure 2: Scalability test of Multi-GPU implementation of NUMA: Weak Scalability of NUMA (running explicit time-steps) for up to 16384 GPUs on the Titan supercomputer is shown. Each node of Titan contains a Tesla K20 GPU. The test is conducted using the unified implementation of CG and DG with and without the overlapping of communication with computation. DG benefits from overlapping of communication with computation as the figure clearly shows; overlapping helps by hiding latency of communication between the CPU and GPU. More details can be found in [5].



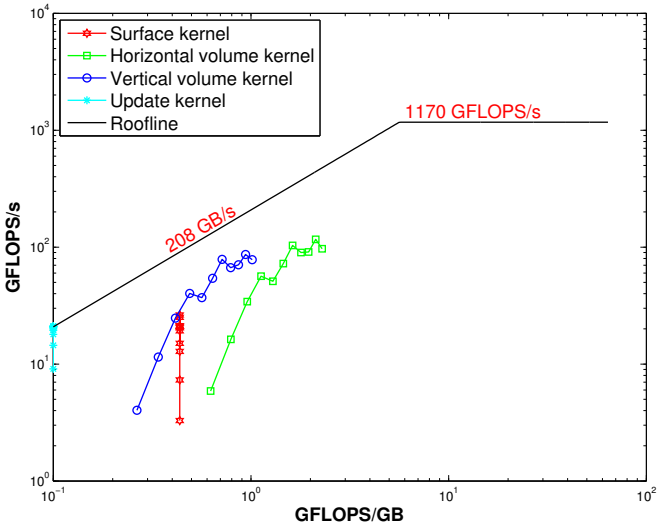
(a) Single precision, continuous Galerkin kernels performance



(b) Double precision, continuous Galerkin kernels performance



(c) Single precision, discontinuous Galerkin kernels performance



(d) Double precision, discontinuous Galerkin kernels performance

Figure 3: Performance of individual kernels in gNUMA. The FLOPS and byte for each kernel are counted manually. Here the single node performance is given in a roofline plot for the performance of the kernels on a Tesla K20c. The roofline performance model give a bound of kernel performance based on bytes read per floating-point operation. More details can be found in [5].

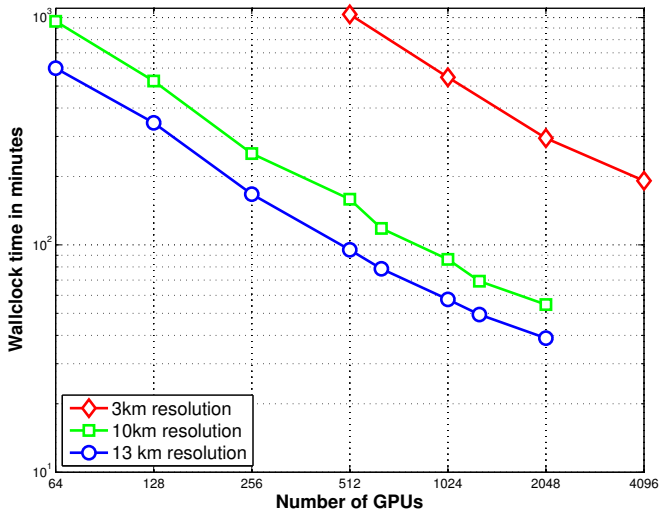


Figure 4: Strong scalability tests on a global scale NUMA simulation using explicit time stepping and grid resolutions of 13 km (6 × 112 × 112 × 4 elements at N=7), 10 km (6 × 144 × 144 × 4 elements at N=7) and 3 km (6 × 448 × 448 × 4 elements at N=7). The number of GPUs required to fulfill a target 100 minutes wallclock time for a one day forecast is about 512, 1024 and 8192 GPUs for the 13km, 10km, and 3km resolutions, respectively.

requirement of NOAA with a modest number of GPUs.

NUMA on Intel's Knights Landing Xeon Phis

In addition to running NUMA on Nvidia hardware, we also conducted tests on Intel's Xeon Phi 'co-processor' Knights Landing (KNL); the KNL is more like a standard CPU with many cores that allows threaded programs to run without code modifications. Because we used OCCA for programming both the CPU and the GPU, we did not need to rewrite the GPU kernels for KNL. However, we added vectorization support for the KNL that was not fundamentally necessary for the GPU. Moreover, the way OCCA kernels are run on the CPU is through what is known as an element-per-node approach where one or more elements are processed with a single OpenMP thread, while a node-per-thread approach is used on the GPU. This design yields optimal performance on both the CPU and GPU. The KNL has 64 or more cores that are able to execute 4 threads per core simultaneously.

In Figs. 5a - 5b, we show the GFLOPS/s (arithmetic compute-intensity) and GB/s (memory bandwidth) of our kernels on the KNL using the OpenMP translation of the OCCA kernels. Due to the lack of tools for evaluating individual kernel performance on the KNL, we use the 'hand-counting' approach for measuring the number of FLOPS performed and bytes transferred per second. As expected, the volume and diffusion kernels show the highest rates of floating point operations at about 210 GFLOPS/s, which is about 7% of peak performance in double precision (3 TFLOPS). The IMEX time-integration update kernel shows maximum bandwidth usage along with other similar kernels such as the No Schur form right hand side evaluation kernels that are bottlenecked by the data transfer rate. The bandwidth usage differs significantly depending on the type of RAM used for storing the data, for which two types are available on the KNL. The

Table 2: Knights Landing (KNL) test: 64 physical cores with upto 4 hyper-threads per core. Time-to-solution of a 3D rising thermal bubble problem is given in seconds. The optimal number of threads for NUMA is running 2 tasks (threads or processes) per core – 128 tasks overall on a single KNL processor. No difference in performance is observed between threads and cores.

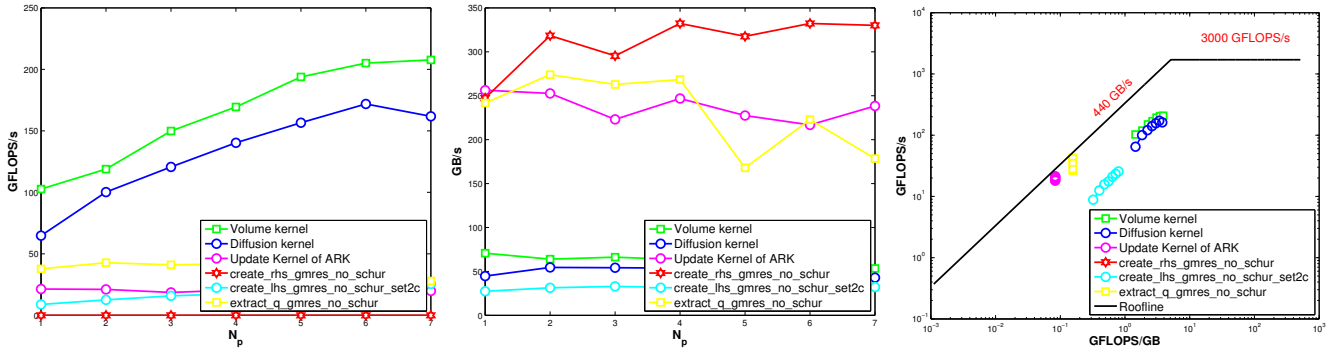
Processes	Threads per process								
	1	2	4	8	16	32	64	128	256
128	83.2	96.82	100						
64	101	82.4	94.94						
32	184	101	81.72	93.7					
16			101	82.49	93.88	97			
8				101	82.3	94			
4					102	84	98.17		
2						104	86.44	102	
1							107	90.7	110

high-bandwidth RAM (440 GB/s MCDRAM) of the KNL is about 5X faster than the standard DRAM4 (90 GB/s peak). The results shown here use all of the 16GB MCDRAM in cache mode. This means we did not need to modify our code, however, better results could potentially be obtained by managing memory ourselves. We can see that the time update kernel, the No Schur form right-hand side and result 'extraction' kernels get close to peak bandwidth usage using the cache mode while most other kernels are far from reaching peak. Looking at the roofline plots, most kernels are memory-bound with the volume and diffusion kernels close to being compute bound.

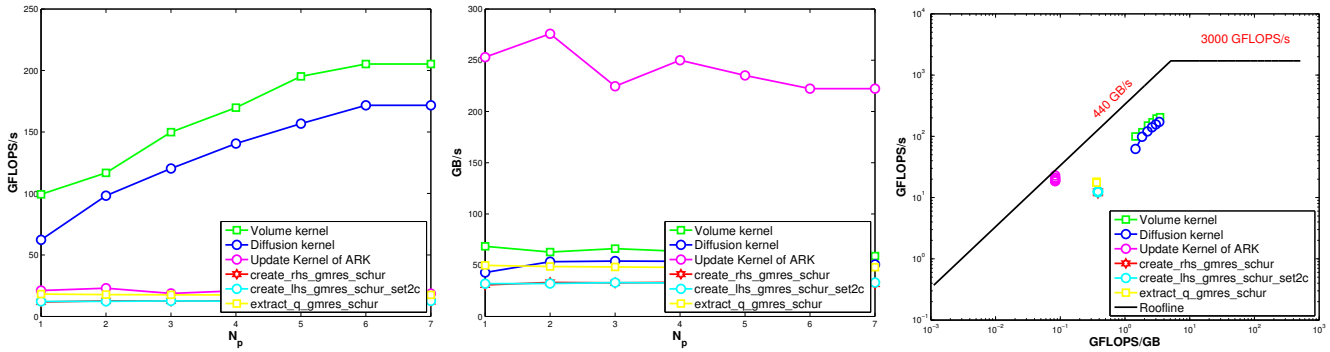
One can run on the KNL with a standard MPI program without OpenMP; however, a hybrid MPI-OpenMP program could perform better as threads could be more 'lightweight' than processes and also because communication between threads is faster through shared memory. The results in Table 2 show that there is little difference between using threads or processes on the Knights Landing (KNL) – which was somewhat a surprise to us as we hoped to gain some benefit from adding OpenMP support. The least amount of time-to-solution is obtained on the diagonal of the table where 128 tasks are launched per KNL device. Deviating from this optimum by using either 1 thread per core or 3 threads per core results in a significant increase in time-to-solution. The largest time-to-solution obtained using 128 MPI processes is about 7% slower than one using purely OpenMP with 128 threads. More details on running NUMA using implicit time-integration on either Nvidia GPUs or Intel's KNL can be found in [4].

Implicit-Explicit NUMA Dynamics on Many-core Computers

High order ARK IMEX methods can be used to improve the accuracy of the results especially when a large Courant number (larger time step) is used. We can see in the bottom left plot of Fig. 7 that the higher order ARK3, ARK4 and ARK5 still give significant speedups over using an explicit RK method. The value of using high-order IMEX methods is that since the time-error is proportional to $\mathcal{O}(\Delta t^k)$ where k is the order of the time-integrator, then we will incur smaller errors while using large time-steps when k is sufficiently large.



(a) 3D-IMEX in No Schur form kernels performance



(b) 3D-IMEX in Schur form kernels performance

Figure 5: Performance of NUMA kernels on the KNL are shown in-terms of GFLOPS/s, GB/s and roofline plots to illustrate their efficiency. The test is done with MCDRAM mode set as cache.

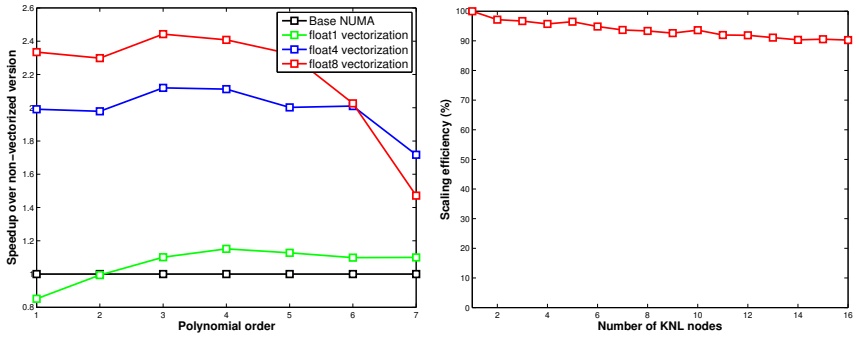


Figure 6: (left) NUMA speedup from vectorization is shown using float1/float4/float8 groups. (right) Strong scaling using 16 KNL nodes in which each KNL node runs 64x2=128 tasks. The problem size for this test is about 32 million nodes.

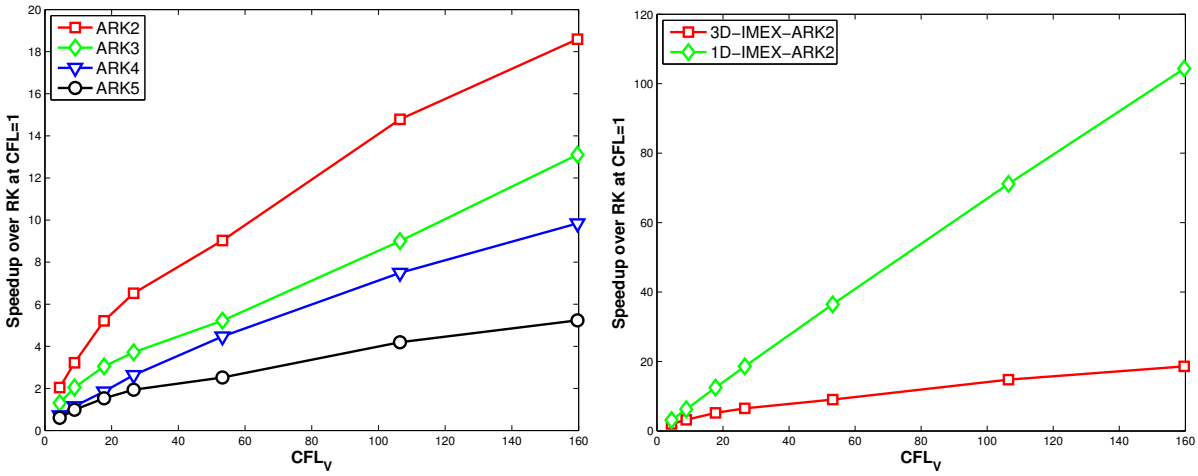


Figure 7: The relative speedup of IMEX over RK for NUMA at different Courant numbers is shown for the acoustic wave problem solved on a cubed sphere grid of $6 \times 10 \times 10 \times 3$***. At*** $C_V = 150$ ***(Courant number with respect to the vertical direction), a relative speedup of about 20X is observed using the second order Additive Runge Kutta (ARK2) scheme. If we, instead, use the 1D-IMEX Schur form with a direct solver, we get a speedup of upto 100X.***

The scalability of 3D-IMEX time-integration on a multi-GPU cluster is evaluated using grid resolutions of 13 km, 10 km and 3 km achieved using 6x112x112x4, 6x144x144x4 and 6x448x448x4 elements, respectively, at polynomial order of 7th degree [6]. We can see in the scalability plot in Fig. 8, that the IMEX method shows a scalability that is as good as the explicit method for all resolutions, while achieving about a 5X relative speedup. IMEX achieves target wallclock minutes at fewer number of GPUs than the explicit time integration method. For example, on the coarse grids, IMEX requires about 64 GPUs to achieve a target 100 minutes wallclock time while RK requires 512 GPUs for the same target; IMEX brings down the wallclock minutes below 10 minutes if we use the same number of GPUs for both, i.e. 512. Therefore, we can conclude that IMEX plays an important role in decreasing both the wallclock minutes on a single-node machine and also the number of nodes required to achieve a specified target wallclock minutes on multi-node clusters. Although this result is well-known on CPU computers, it is not as obvious on many-core computers, particularly for GPUs. There exist very few works in the literature on porting IMEX methods to the GPU - to our knowledge the work described here is the first time that fully 3D-IMEX has been implemented on many-core for an atmospheric model.

[6] The notation $6 \times Ne_\xi \times Ne_\eta \times Ne_\zeta$ denotes 6 panels of the cubed sphere grid with $Ne_\xi \times Ne_\eta$ elements on each panel with Ne_ζ elements in the radial direction.

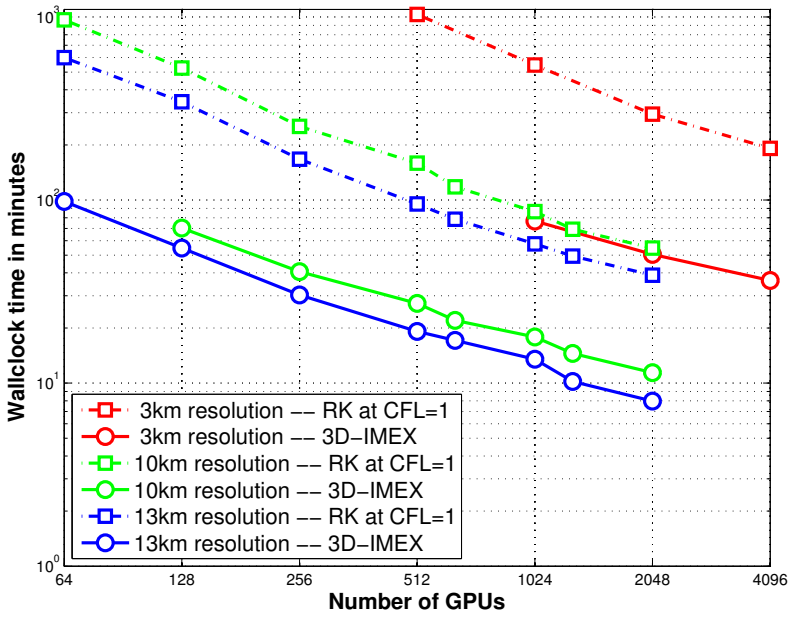


Figure 8: Strong-scalability of 3D-IMEX NUMA on multiple GPUs for grid resolutions of 13 km ($6 \times 112 \times 112 \times 4$ ***elements at N=7), 10 km*** ($6 \times 144 \times 144 \times 4$ ***elements at N=7) and 3 km*** ($6 \times 448 \times 448 \times 4$ ***elements at N=7).***

LOO.PY and Semi-Auto-Tuned Kernels

Next we provide some results published in [8] about using Loo.py to transform generic Fortran code into kernels (similar to the hand written kernels in NUMA) for use with methods for solving Euler's equations. For example, here is the code for the volume derivative in the r element reference direction:

```
subroutine refFluxR(Ne, Nq, geo, D, q, rhsq, p_p0, &
    p_Gamma, p_R)
  implicit none
  integer*4 Ne, Nq, e, i, j, k, n
  real geo(Nq,Nq,Nq,11,Ne), D(Nq,Nq)
  real rhsq(Nq,Nq,Nq,8,Ne), q(Nq,Nq,Nq,8,Ne)

  real U1, U2, U3, Rh, Th, P, Q1, Q2, Q3
  real U1flx, U2flx, U3flx, Rhflx, Thflx
  real Q1flx, Q2flx, Q3flx, p_p0, p_Gamma, p_R
  real g11, g21, g31, Jinv, udotGradR, JiD

  do e = 1, Ne
    do k = 1, Nq
      do j = 1, Nq
        do i = 1, Nq
          do n = 1, Nq
!$loopy begin tagged: local_prep
            U1 = q(n,j,k,1,e)
            U2 = q(n,j,k,2,e)
            U3 = q(n,j,k,3,e)
            Rh = q(n,j,k,4,e)
            Th = q(n,j,k,5,e)
            Q1 = q(n,j,k,6,e)
            Q2 = q(n,j,k,7,e)
            Q3 = q(n,j,k,8,e)

            g11 = geo(n,j,k,1,e)
            g21 = geo(n,j,k,2,e)
            g31 = geo(n,j,k,3,e)

            Jinv = geo(i,j,k,10,e)

            P = p_p0*(p_R*Th/p_p0) ** p_Gamma
            udotGradR = (g11*U1 + g21*U2 + g31*U3)/Rh
!$loopy end tagged: local_prep

            JiD = Jinv*D(i,n)

!$loopy begin tagged: compute_fluxes
            U1flx = U1*udotGradR + g11*P
            U2flx = U2*udotGradR + g21*P
            U3flx = U3*udotGradR + g31*P
            Rhflx = Rh*udotGradR
            Thflx = Th*udotGradR
            Q1flx = Q1*udotGradR
            Q2flx = Q2*udotGradR
            Q3flx = Q3*udotGradR
```

Nq	Ne	Wall Time (ms)	GFLOP/s	Bandwidth (GBs^{-1})
			Radeon R9 FURY X	
4	55300	3.20	206	168
8	6910	1.36	816	395
12	2050	4.26	367	126
16	864	6.46	312	83.3
			GeForce GTX TITAN X	
4	55300	1.78	369	302
8	6910	2.07	538	260
12	2050	2.21	709	244
16	864	2.12	953	254
				Tesla K40c
4	55300	5.54	119	97.1
8	6910	3.29	338	164
12	2050	5.35	293	101
16	864	5.54	364	97.1

Table 3: Performance of the Loo.py optimized horizontal volume kernel for different numbers of per-element grid points. Note, the total number of DG grid points is equal between runs. The FLOP/s ***and bandwidth are estimated, not measured. Here there are*** `Ne` ***elements with*** $\texttt{Nq}^3$ ***points per element.***

```
!$loopy end tagged: compute_fluxes

          rhsq(i,j,k,1,e) = rhsq(i,j,k,1,e) - JiD*U1flx
          rhsq(i,j,k,2,e) = rhsq(i,j,k,2,e) - JiD*U2flx
          rhsq(i,j,k,3,e) = rhsq(i,j,k,3,e) - JiD*U3flx
          rhsq(i,j,k,4,e) = rhsq(i,j,k,4,e) - JiD*Rhflx
          rhsq(i,j,k,5,e) = rhsq(i,j,k,5,e) - JiD*Thflx
          rhsq(i,j,k,6,e) = rhsq(i,j,k,6,e) - JiD*Q1flx
          rhsq(i,j,k,7,e) = rhsq(i,j,k,7,e) - JiD*Q2flx
          rhsq(i,j,k,8,e) = rhsq(i,j,k,8,e) - JiD*Q3flx
        end do
      end do
    end do
  end do
 end do
end subroutine refFluxR
```

Using the transformations detailed in [8], this Fortran code is used to generate a horizontal volume kernel similar to the hand-written one in NUMA. We assessed its performance on various GPUs and present the results in Table 3. From this assessment we find that there is a sensitivity of the performance based on the number of points in a given direction of the element, `Nq`.

IMPACT/APPLICATIONS

Ensuring that the U.S. gains and maintains a strategic advantage in medium-range weather forecasting requires pooling knowledge from across the disparate U.S. government agencies currently involved in both climate and weather prediction modeling. The new computer architectures currently coming into maturity have leveled the playing field because only those that embrace this technology and fully commit to harnessing its power will be able to push the frontiers of atmosphere-ocean modeling beyond its current state. The work in this project is critical to developing and distributing the knowledge of accelerator-based computing that will support the use of the new platforms in many of the climate, weather, and ocean laboratories around the country.

TRANSITIONS

Improved algorithms for model processes will be transitioned to 6.4 as they are ready, and will ultimately be transitioned to FNMOC.

RELATED PROJECTS

The Earth System Modeling Framework (ESMF) together with the NUOPC Interoperability Layer form the backbone of the Navy ESPC software coupling infrastructure. We will enable the many-core mini-apps and NUMA to be used as components in the Navy ESPC by implementing them as a NUOPC compliant ESMF components. This will bring our work the ESPC community enabling coupling to codes from other projects such as HYCOM and Wavewatch III.

REFERENCES

[1] J. F. Kelly and F. X. Giraldo. "Continuous and discontinuous Galerkin methods for a scalable three-dimensional nonhydrostatic atmospheric model: Limited-area mode". English. In: *Journal of Computational Physics* 231.24 (Oct. 2012), pp. 7988–8008. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2012.04.042.

[2] S. Swaminarayan. *Exaflops, petabytes, and gigathreads... oh my!* DoE Advanced Scientific Computing Research (ASCR) 2011 Workshop on Exascale Programming Challenges. 2011. URL: http://science.energy.gov/~/media/ascr/pdf/research/cs/Programming_Challenges_Workshop/Siriam%20Swaminarayan.pdf.

PUBLICATIONS

[3] D. S. Abdi and F. X. Giraldo. "Efficient construction of unified continuous and discontinuous Galerkin formulations for the 3D Euler equations". In: *Journal of Computational Physics* 320 (2016), pp. 46–68. DOI: 10.1016/j.jcp.2016.05.033. [published, refereed].

[4] D. S. Abdi, F. X. Giraldo, E. M. Constantinescu, L. E. Carr III, L. C. Wilcox, and T. Warburton. *Acceleration of an implicit-explicit non-hydrostatic unified model of the atmospheric (NUMA) on manycore processors.* 2016. [submitted].

[5] D. S. Abdi, L. C. Wilcox, T. Warburton, and F. X. Giraldo. *A GPU accelerated continuous and discontinuous Galerkin non-hydrostatic atmospheric model*. July 2016. [submitted].

[6] A. Klöckner. "Loo.py: from Fortran to performance via transformation and substitution rules". In: *Proceedings of the 2nd ACM SIGPLAN International Workshop on Libraries, Languages, and Compilers for Array Programming*. ARRAY'15. Portland, OR, USA: ACM, 2015, pp. 1–6. ISBN: 978-1-4503-3584-3. DOI: 10.1145/2774959.2774969. [published, refereed].

[7] A. Klöckner. "Loo.py: transformation-based code generation for gpus and cpus". In: *Proceedings of ACM SIGPLAN International Workshop on Libraries, Languages, and Compilers for Array Programming*. ARRAY'14. Edinburgh, United Kingdom: ACM, 2014, 82:82–82:87. ISBN: 978-1-4503-2937-8. DOI: 10.1145/2627373.2627387. [published, refereed].

[8] A. Klöckner, L. C. Wilcox, and T. Warburton. "Array program transformation with Loo.Py by example: high-order finite elements". In: *Proceedings of the 3rd ACM SIGPLAN International Workshop on Libraries, Languages, and Compilers for Array Programming*. ARRAY 2016. Santa Barbara, CA, USA: ACM, 2016, pp. 9–16. ISBN: 978-1-4503-4384-8. DOI: 10.1145/2935323.2935325. [published, refereed].

[9] A. Müller, M. A. Kopera, S. Marras, L. C. Wilcox, T. Isaac, and F. X. Giraldo. *Strong scaling for numerical weather prediction at petascale with the atmospheric model NUMA*. Nov. 2015. [submitted] preprint: http://arxiv.org/abs/1511.01561.