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

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: N0001415WX00694; N0001415WX01603; N000141310873; N000141410117; N0001414AF00002

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://faculty.nps.edu/fxgirald/projects/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.

OBJECTIVES

The objective of this project is fourfold. The *first* objective is to identify the bottlenecks of NUMA and then circumvent these bottlenecks through the use of: (1) analytical tools to identify the most computationally intensive parts of both the dynamics and physics; (2) intelligent and performance portable use of heterogeneous accelerator-based many-core machines, such as General Purpose Graphics Processing Units (GPGPU or GPU, for short) or Intel's Many Integrated Core (MIC), for the dynamics; and (3) intelligent use of accelerators for the physics. The *second* objective is to share and extend the tools we use to develop code that is portable across threading APIs (achieved by using OCCA https://libocca.org) and has portable performance (by using Loo.py http://mathema.tician.de/software/loopy). The features we are adding to OCCA and Loo.py under this project are targeted first to help our many-core acceleration of NUMA effort but should be generally applicable to many scientific computing codes from the general climate, weather, and ocean modeling laboratories around the country. The *third* objective is to implement Earth System Modeling Framework (ESMF) interfaces for the accelerator-based NUMA mini-apps allowing the study of coupling many-core based components. We will investigate whether the ESMF data structures can be used to streamline the coupling of models in light of these new computer architectures which require memory access that has to be carefully orchestrated to maximize both cache hits and bus occupancy for out of cache requests. The *fourth* objective is to implement NEPTUNE (Navy Environmental Prediction sysTem Using the NUMA corE) as an ESMF component allowing NEPTUNE to be used as an atmospheric component in a coupled earth system application. A specific outcome of this objective will be a demonstration of a coupled air-ocean-wave-ice system involving NEPTUNE (with NUMA as its dynamical core), HYCOM, Wavewatch III, and CICE within the Navy ESPC. The understanding gained through this investigation will have a direct impact on the Navy ESPC that is currently under development.

APPROACH

Following the lead of various DoE labs [4], we are adapting NUMA to accelerator-based many-core machines in a step-by-step process to achieve our first objective. At each step we are developing *mini-apps* which are self-contained programs that capture the essential performance characteristics of different algorithms in NUMA. Note that NUMA contains both continuous Galerkin (CG) and discontinuous Galerkin (DG) modes for discretization. This plan to partition the development of the heterogeneous architecture version of NUMA into small chunks of work that can be handled somewhat independently will allow us to produce (at every stage of the work pipeline) a result that is beneficial not only to the NUMA developers and user groups but also to the larger climate, weather, and ocean modeling community. The many-core mini-apps that will be developed will include:

Dynamics

Explicit-in-time CG a continuous Galerkin discretization of the compressible Euler mini-app with explicit time integration;

Explicit-in-time DG a discontinuous Galerkin discretization of the compressible Euler mini-app with explicit time integration;

Vertically Semi-Implicit CG a continuous Galerkin discretization of the compressible Euler mini-app with vertically implicit semi-implicit time integration;

Vertically Semi-Implicit DG a discontinuous Galerkin discretization of the compressible Euler mini-app with vertically implicit semi-implicit time integration;

Physics

Moisture a parameterized moisture mini-app; and

Once the performance of a mini-app is accepted it will be considered for adoption into NUMA. Placing the kernels back into NUMA is being lead by Giraldo and his postdoctoral researcher Abdi. We will also make these mini-apps available to the community to be imported into other codes if desired. Wilcox is working closely with Warburton and his team to lead the effort to develop the mini-apps including hand rolled computational kernels optimized for GPU accelerators. These kernels are written “hand-written” in OCCA, a library Warburton’s group is developing that allows a single kernel to be compiled using many different threading frameworks, such as CUDA, OpenCL, OpenMP, and Pthreads. We are initially developing hand-written kernels to provide a performance target for the Loo.py generated kernels. Parallel communication between computational nodes will use the MPI standard to enable the mini-apps to run on large scale clusters. Using these community standards for parallel programing will allow our mini-apps to be portable to many platforms, however the performance may not be portable across devices. For performance portability, we, lead by Klöckner, are using Loo.py to generate OCCA kernels which can be automatically tuned for current many-core devices along with future ones.

The second objective is to expand (as needed by the mini-apps) the OCCA and Loo.py efforts, lead by Warburton and Klöckner. These will be extended naturally in tandem with the requirements that emerge during the development of the mini-apps. We will take a pragmatic approach where features will only be added as they are needed by the mini-apps or will aid in the transition of the mini-app technology back into NUMA. For the sharing part of the objective, both OCCA and Loo.py are open source and can be downloaded from https://github.com/libocca/occa and https://github.com/inducer/loopy, respectively. They are operational and are ready to be evaluated for use in other projects. As it warrants, we will give presentations and demonstrations of the tools to help increase their adoption.

The third objective, lead by Campbell, is to implement Earth System Modeling Framework (ESMF) interfaces for the accelerator-based computational kernels of NUMA allowing the study of coupling many-core based components. Once we have working mini-apps implementing a significant set of physics, we will coordinate with the "An Integration and Evaluation Framework for within the proposed ESPC Coupling Testbed" if it is deemed useful to have the mini-app to help test coupling of many-core components.

The fourth goal, lead by Campbell, is to implement NEPTUNE as an ESMF component allowing NEPTUNE. This will be done in collaboration with the developers of NEPTUNE at NRL Monterey. Once NEPTUNE has been made a component, we will move to running the coupled air-ocean-wave-ice system involving NEPTUNE (with NUMA as its dynamical core), HYCOM, Wavewatch III, and CICE within the Navy ESPC.

WORK COMPLETED

In the course of this project (the first three years plus the two year option), we plan to carry out the following work items:

1. identify current bottlenecks in the NUMA modeling system;
2. port the explicit time-integration portion of the dynamics onto many-core devices;
3. port the moisture schemes onto many-core devices;
4. port the implicit-in-the-vertical dynamics onto many-core devices;
5. port relevant physics onto many-core devices;
6. transition many-core kernels into NUMA;
7. adapt the Loo.py code generator for the needs of the project;
8. develop a source-to-source translation capability built on Loo.py to facilitate the NUMA transition;
9. implement ESMF interfaces for many-core components;
10. implement NEPTUNE as an ESMF component;
11. assess performance against current modeling suite.

The management plan for these work items is shown in table 1. Below is a summary on the progress we have obtained on these work items.

Identifying bottlenecks of NUMA This year we have completed two major tasks in identifying bottlenecks of NUMA. The first was the incorporation of `p4est`[1] into NUMA to enable efficient mesh generation on large distributive memory machines. The second was to unify the CG and DG NUMA Fortran codes in preparation for using the kernels developed in the mini-apps.

[1]`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	37–48	49–60
Identifying bottlenecks		▷				
Explicit dynamics		▷	▷			
Moisture		▷	●	●		
Vertically-implicit dynamics			▷	●	●	
More physics processes					●	●
Transition many-core kernels into NUMA			▷	●	●	●
Adapt Loo.py		▷	▷	●	●	
Develop source translation tool			▷	●	●	●
Implement ESMF in mini-apps				●	●	●
Implement ESMF in NEPTUNE		▷		●		
Assess Performance		▷	▷	●	●	●
Disseminate work (Publications)			▷	●	●	●

Table 1: Time-line of proposed activities for different months of the project. The triangle symbol indicates activities that have started.

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 Dr. Tobin Isaac and other members of the NUMA team) incorporated `p4est` into NUMA.

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 (call it `numa3dDG`) has been folded back into the master branch of NUMA. This new version of NUMA is capable of handling two data layouts for CG and one for DG — similar to the unified construction of `numa2dCGDG_AMR`, the 2D adaptive mesh refinement version of NUMA. This unification in the 3D code will now bring the NUMA code and the mini-apps closer together.

One of the next set of goals of the project is to combine the MPI communication for both CG and DG. Also, the CG-MPI implementation does not currently use computation-communication interleaving. This, however, exists in the DG-MPI implementation which effectively "hides" the communication cost. We have sketched out a similar idea for CG which will make the CG MPI-implementation faster. We will begin testing this approach this fiscal year. The idea behind this approach is as follows: perform the volume integration of internal nodes while communication is done on boundary nodes. The DG MPI-implementation communicates across boundary faces first (using non-blocking sends/receives) and, while the messages are sent and received, the volume integrals are computed. This approach effectively hides the communication costs and is one of the reasons why our DG solvers scale so well as we have shown in previous publications (e.g., Kelly and Giraldo [2]).

Explicit Dynamics During this quarter, steady progress has been made on the Euler mini-app, `gNUMA`. This includes the debugging of discontinuous diffusion kernels. Thus, we now have all the basic explicit dynamics implemented from NUMA. We have an explicitly time-stepped continuous/discontinuous Galerkin Euler solver with viscosity based stabilization.

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`[2] 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. The plan is to reproduce all of the functionality of `gNUMA` in `bigNUMA`. We are using a 2D isentropic vortex benchmark to check for correctness. We will add the rest of the benchmarks as things progress.

Working with NPS Assistant Professor Jeremy Kozdon (one of the authors of `bfam`) we have developed the infrastructure needed to build *hp*-adaptive DG discritizations using OCCA on multiple devices using MPI. This includes dynamic adaptivity.

The next milestones for `bigNUMA` are:

- fold in adaptive infrastructure developed with Kozdon;
- enable CG discretizations on multiple devices;
- enable CG discretizations on adaptive grids;
- moisture physics;
- (semi-)implicit time-stepping; and
- add scalable asynchronous I/O.

Vertically-Implicit Dynamics The work completed so far uses explicit time stepping methods; however; 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. We plan to port NUMA's 1D IMEX time integration schemes to OCCA in the next year.

Moisture In our explicit dynamics mini-app we have designed the kernels for 2–3 moisture variables to include warm moist processes in the model. We are now determining which moisture parameterization to use. We are also considering using the warm air moist processes described in Satoh [3]. To verify the implementation we are going to start with the moist bubble testcase from Duarte, Almgren, Balakrishnan, Bell, and Romps [1].

OCCA A new source-to-source kernel translator has been added to the OCCA portability run-time library. The kernel translator includes a prototype module that can translate native OpenCL and CUDA kernel source code into the OCCA intermediate kernel language. This adds additional use cases for OCCA including for instance the ability to deploy CUDA kernels on compute devices that do not natively support NVIDIA's proprietary kernel language.

[2] `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/

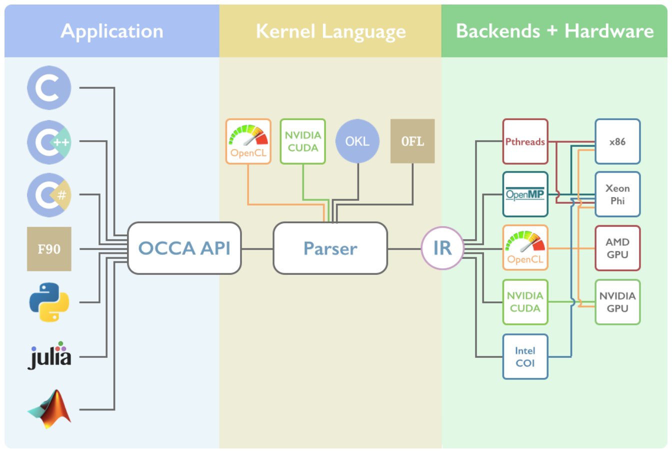

Figure 1: Overview of OCCA API, kernel languages, and programming model support. See libocca.org for tutorials and documentation

The flexible kernel translator has also enabled us to create new kernel languages. We have developed the OCCA Kernel Language (OKL) that lets programmers write C-style kernel code that looks in essence like serial code. With minimal code decoration the kernels can be translated into thread parallel code that can be executed using CUDA, OpenCL, or OpenMP. Additionally OCCA can now wrap arrays provided by these APIs improving interoperability.

Given the preference of the climate modeling community for Fortran based codes we also added support for the new OCCA Fortran kernel Language (OFL) that provides the same features as OKL but with F90 style syntax. There are now two types of possible Fortran programmers that can use OCCA:

Expert Use the F90 interface to the OCCA libraries to execute OFL Fortran style kernels on a wide range of devices (CPU, GPU, Accelerators). Currently the onus is on the programmer to schedule host↔device data movement, and also to carefully construct the OFL kernel implementations to map efficiently to the hardware.

Intermediate Use the F90 OCCA interface to interact with the OCCA libraries and express the kernels in the Floo.py Fortran based kernel language. The programmer can add kernel tuning commands to the Floo.py kernels with Python based code that applies code transformations useful for optimization. The combination of OCCA and the Floo.py library offers the programmer a way to investigate the possibility of performance portability.

There is still a gap in the OCCA support for Fortran programmers who do not have accelerator experience. Activities at Rice/Virginia Tech are geared towards making multithreaded programming as simple as possible. The OCCA library currently in the planning stages will provide optional support for automated data movement and for kernel optimization using source code analysis and run-time detective work.

In this quarter the Rice/Virginia Tech team has 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++, C#, Python, Julia, and MATLAB. These interfaces have been complemented with a native F90 front end 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. In this quarter we have made steps towards making the thread programming model much simpler. We have developed OKL and OFL kernel languages based on C and Fortran respectively. The kernel syntax allows kernel development using source code that is minimally different from regular C or Fortran. "For loops" are lightly annotated to indicate that they should be treated as parallel for loops with no loop carried dependencies.

The following is the taxonomy of options for a programmer to develop many-core code using OCCA:

Expert Use the F90 interface to the OCCA libraries to execute OFL Fortran style kernels on a wide range of devices (CPU, GPU, Accelerators). The expert will schedule movement of data between HOST and DEVICE and also carefully construct the OFL kernel implementations to map efficiently to the hardware. In this mode OCCA does not make any independent steps on data movement or on which loops to treat as parallel.

Intermediate Use the F90 OCCA interface to interact with the OCCA libraries and express the kernels in the loopy Fortran based kernel language. The programmer can add kernel tuning commands to the Floo.py kernels with Python based code that applies code transformations useful for optimization. The combination of OCCA and the loopy library offers the programmer a way to investigate the possibility of performance portability.

Beginner In this quarter we have added support for universal virtual addressing (UVA) to the C and C++ interfaces. This provides the option to use memory pointers instead of OCCA memory objects. In addition we have developed kernel code analysis so that the beginner developer can rely on OCCA to automatically migrate data between the HOST and DEVICE for arrays that are allocated as managed memory UVA arrays. An example here:

```
// managed alloc of HOST+DEVICE arrays
float *a = (float*) device.managedUvaAlloc(entries * sizeof(float));
float *b = (float*) device.managedUvaAlloc(entries * sizeof(float));
float *ab = (float*) device.managedUvaAlloc(entries * sizeof(float));

// HOST initialization
for(int i = 0; i < entries; ++i){
  a[i] = i;
  b[i] = 1 - i;
  ab[i] = 0;
}

// queue DEVICE kernel
addVectors(entries, a, b, ab);

// make sure DEVICE has finished
device.finish();
```

```
// HOST outputs results
for(int i = 0; i < 5; ++i)
  std::cout << i << ": " << ab[i] << '\n';
```

shows how much this can simplify the process of writing the driver application. We are currently planning steps to bring this capability to the F90 OCCA API.

To simplify the process of porting code to OCCA we are actively developing the OCCA automagic kernel language OAK and Fortran variant OAF. We are building a kernel analysis tool that detects which loops can be labelled as parallel loops, i.e., automatically ports a tightly described set of C-like serial code to the OKL syntax. The current barebones OAK parser can automatically label serial versions of the kernels developed for the `gNUMA` simulation tool developed for this project.

On the OCCA front, general development has proceeded. A few minor bugs have been found and fixed as a result of the fact that we are now using the Fortran interface in NUMA. We expect this to continue as we move more features back into NUMA.

Transition many-core kernels into NUMA The first stage of unification of the CG and DG codes has been completed. There is now 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 code has begun. We first started by running benchmarks in `gNUMA`. This development reuses the kernels from the mini-apps `gNUMA` and `bigNUMA` as well as the Fortran 90 interface to OCCA developed under this project. The goal of this effort is to bring the OCCA version of NUMA into a form that can be transferred to NRL to be run in NEPTUNE.

With regard to scalability, we would like to rewrite CG in such a way that we can do communication-computation overlap so that it will see the same benefit as DG under weak-scaling. In addition, we would like to implement and test direct GPU to GPU communication to reduce communication latency.

Adapt Loo.py and develop source translation tool The objective for this part of the work is 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. 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. It is expected that the created transformation machinery will generalize and, going forward, will allow an easy route towards high performance via streamlined experimentation and autotuning. The latter is particularly promising because it is trivial to create based on the infrastructure accumulated during this cycle.

- The programmer interface of the source translation tool was modified to put the transformation code in charge of the entire translation pipeline, from preprocessing to parsing to transformation. This enables many advanced use cases that include transformations that work across multiple kernels.

- One example of such a transformation that a conventional compiler cannot achieve is the fusion of multiple kernels into a single loop structure. We have successfully demonstrated this transformation on one of the most performance-sensitive parts of gNUMA.
- A larger cross-section of the Fortran standard is now supported by the translation tool, including conditional statements.
- Fortran parsing and translation is now more tightly integrated with the remainder of the translation tool. Automated tests verify the functionality of the tool.
- A build cycle was established that creates a standalone binary of the translation tool. This makes it possible for researchers to try out the tool by downloading one single file, which is immediately runnable. To make this possible, the tool itself was modified to have fewer mandatory dependencies.
- Work continues towards making the performance-critical parts of the `gNUMA` code entirely transformation based, for easy performance portability and optimization experimentation.
- A novel mechanism was created that allows the transformation code to refer to segments of the source to be transformed with the help of so-called tags.
- Many advanced transformations were newly implemented, such as affine loop transformations, buffering of off-chip variables in on-chip memory across complicated loop nests,
- Begin work on an automatic, self-calibrating performance model for Loo.py kernels with graduate student James Stevens.
- Presented an article [5] on Fortran translation and transformation via substitution rules at 2015 ARRAY workshop.

Implement ESMF in NEPTUNE There is nothing to report on this front right now.

Assess Performance We are continuously assessing performance of all of the different strategies for running code on many-core processors. The main assessment this year was of the occa kernels in the NUMA code itself detailed in the RESULTS section below.

Communications with the community At NPS we met with Valentine Anantharaj from ORNL to discuss how we are using OCCA in `gNUMA`. The tools group at ORNL seems interested and is following up with Warbuton's team at Rice/Virginia Tech.

Giraldo held a workshop on "Galerkin methods with applications in weather and climate forecasting" on Mar 23, 2015 – Mar 27, 2015 at ICMS, 15 South College Street, Edinburgh, http://www.icms.org.uk/workshop.php?id=334, to bring together senior and junior international researchers from the national labs, academia, and industry who are actively engaged in the development of Galerkin methods for numerical weather prediction. This was a perfect place to discuss the work of this project. Due to an unfortunate event, Wilcox was unable to attend, however Giraldo presented some of our mini-app work in his place.

Wilcox met Alexander Heinecke from the Intel Parallel Computing Lab at SIAM GS15. We are in discussions with Alexander to help us port our computational kernels to the Xeon Phi. Once we hand roll these kernels we hope to be able to use Loo.py with OCCA to generate these in a programmatic fashion.

Giraldo and a postdoc shared by Giraldo and Wilcox (Abdi) will attend the GPU Hackathon being held at Oak Ridge National Laboratory. The goal in attending this hackathon is to include the standard way currently being used by most atmospheric modelers to port there applications to the GPU, i.e., via OpenACC compiler directives (or pragmas). At these types of workshops, we plan to convey to the community that using other strategies (like OCCA with Floo.py) is no more difficult than restructuring existing code in order to better exploit OpenACC.

RESULTS

NUMA now has 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. 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."[3]

Some of the kernels developed for the mini-apps have been transitioned into NUMA using the OCCA F90 host API. With these kernels incorporated, NUMA is now about to run on many-core devices. So we have tested NUMA in two ways: (1) a weak scaling study on the Titan supercomputing system at Oak Ridge National Laboratory; and (2) a speed comparison of the unified version of NUMA with NUMA using the OCCA kernels.

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 have 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 80% weak scaling efficiency on tens of thousands of GPUs. Different implementations of the unified CG/DG model are tested, where the best performance was achieved by DG with overlapping computation-communication to hide latency.

The accuracy and performance of our OCCA kernels has been verified using several benchmark problems representative of different scales of atmospheric dynamics. We show in Figure 3 a GPU accelerated simulation of an acoustic wave propagation on the sphere solved using explicit time stepping algorithm. Here we present timings for the unified version of NUMA with the original Fortran kernels and the OCCA kernels from the mini-apps using explicit time stepping. It is difficult to compare codes running on the CPU versus those running on the GPU, the best possible

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

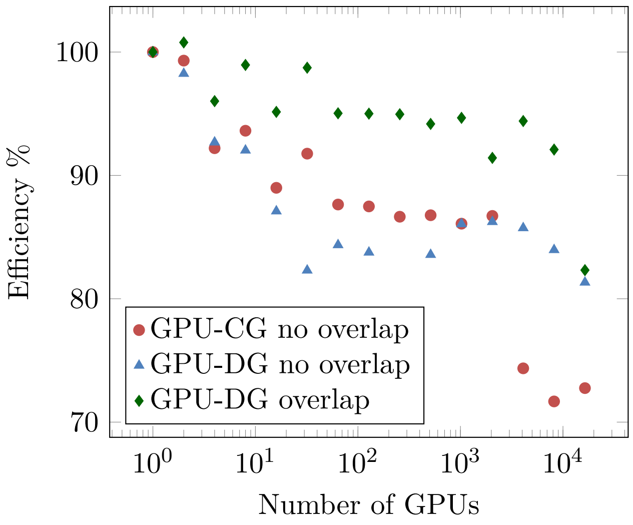

Figure 2: Scalability test of Multi-GPU implementation of NUMA: Weak Scalability of NUMA (running explicit time-steps) for up to 16384 GPUs on the Titan supercomputer is shown. Each node of Titan contains a Tesla K20 GPU. The test is conducted using the unified implementation of CG and DG with and without the overlaping 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.

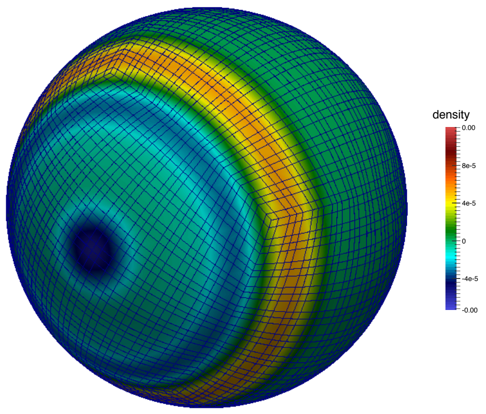

Figure 3: Simulation on the GPU: Acoustic wave propagating on the sphere at the speed of sound is simulated using the GPU on a cubed sphere grid. The result after 5 hours is shown.

Test case	Double precision			Single precision		
	CPU (s)	GPU (s)	Speedup	CPU (s)	GPU (s)	Speedup
2D rising-thermal bubble	930.1	27.8	33.4	612.3	13.4	45.6
3D rising-thermal bubble	4408.9	141.9	31.1	3097.0	54.5	56.8
Acoustic wave on the sphere	3438.8	96.7	35.6	2379.9	44.4	53.6

Table 2: Speedup comparison between a single core CPU run using the Fortran kernels and a GPU run using the OCCA kernels for both single precision and double precision calculations. The GPU card is NVIDIA Tesla C2070 and the CPU is Intel Xeon E5645 @2.4GHZ

Test case	Double precision			Single precision		
	CPU (s)	GPU (s)	Speedup	CPU (s)	GPU (s)	Speedup
2D rising-thermal bubble	930.1	8.87	104.9	612.3	4.67	131.0
3D rising-thermal bubble	4408.9	41.47	106.3	3097.0	18.68	165.8
Acoustic wave on the sphere	3438.8	26.72	128.7	2379.9	15.56	152.9

Table 3: Speedup comparison between a single core CPU run using the Fortran kernels and GPU run using the OCCA kernels for both single precision and double precision calculations. The GPU card is GTX Titan Black and the CPU is Intel Xeon E5645 @2.4GHZ

comparison would probably be solution accuracy per dollar. This dollar dollar includes not only the cost of purchasing the machine the code runs on but running and maintaining it. As one might expect, this is a hard thing to measure. Right now we just want to know what kind of speedups in time to solution we can achieve. In Tables 2–5 we compare the speed of the time-stepping loop for explicit DG NUMA with the Fortran kernels running on a single CPU core against NUMA with the OCCA kernels running on a GPU. In Tables 2–4 this comparison is done for three test cases:

2D rising-thermal bubble a 100 element simulation with polynomial order 7;

3D rising-thermal bubble a 1000 element simulation with polynomial order 5; and

Acoustic wave on the sphere a 1800 element simulation with polynomial order 4.

Table 5 shows the speedup achieved with different mesh configurations for the 2D rising-thermal bubble case. Note that the 2D rising thermal bubble case is run with the 3D version of the code 1 element deep.

Test case	Double precision		
	CPU (s)	GPU (s)	Speedup
2D rising-thermal bubble	20.49	2.79	7.33
3D rising-thermal bubble	89.34	13.15	7.10
Acoustic wave on the sphere	94.14	12.48	7.88

Table 4: Speedup comparison between CPU (on all cores using MPI) and GPU for double precision calculations. The GPU card is K20 and the CPU is 16-core AMD Opteron 6274 @2.2GHz. Here N is the polynomial order of the tensor product basis functions.

N	$10 \times 10 = 100$ elements			$30 \times 30 = 900$ elements			$40 \times 40 = 1600$ elements		
	CPU (s)	GPU (s)	Speedup	CPU (s)	GPU (s)	Speedup	CPU (s)	GPU (s)	Speedup
2	1.46	0.75	1.95	10.62	3.42	3.11	18.83	5.51	3.41
3	2.68	0.87	3.08	22.01	4.32	5.08	41.53	7.08	5.87
4	5.30	1.14	4.64	46.45	6.01	7.73	81.91	10.03	8.17
5	8.12	1.60	5.05	77.03	11.27	6.83	137.49	19.51	7.05
6	13.89	2.40	5.78	122.27	17.80	6.86	210.35	31.06	6.77
7	20.49	2.79	7.33	195.61	21.16	9.24	343.74	36.89	9.32

N	$80 \times 80 = 6400$ elements			$120 \times 120 = 14400$ elements			$160 \times 160 = 25600$ elements		
	CPU (s)	GPU (s)	Speedup	CPU (s)	GPU (s)	Speedup	CPU (s)	GPU (s)	Speedup
2	80.72	19.83	3.92	184.19	43.96	4.19	336.19	77.90	4.31
3	179.07	25.97	6.65	405.15	57.81	7.00	729.17	102.21	7.13
4	350.54	37.69	9.09	798.50	84.04	9.50	1392.60	148.86	9.36
5	587.17	75.97	7.73	1329.79	170.02	7.82	2352.46	301.78	7.79
6	925.25	121.93	7.59	2086.84	273.18	7.64			
7	1406.61	144.79	9.71	3158.43	324.89	9.72			

Table 5: 2D rising thermal bubble: Speedup comparison between CPU (on all cores using MPI) and GPU for double precision calculations at different number of elements and polynomial orders. The GPU card is K20 and the CPU is 16-core AMD Opteron 6274 2.2GHz. Here N is the polynomial order of the tensor product basis functions.

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] M. Duarte, A. S. Almgren, K. Balakrishnan, J. B. Bell, and D. M. Romps. "A numerical study of methods for moist atmospheric flows: compressible equations". In: *ArXiv e-prints* (Nov. 2013). arXiv: 1311.4265 [physics.ao-ph].

[2] 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.

[3] M. Satoh. "Conservative scheme for a compressible nonhydrostatic model with moist processes". In: *Monthly Weather Review* 131.6 (June 2003), pp. 1033–1050. DOI: 10.1175/1520-0493(2003)131<1033:CSFACN>2.0.CO;2.

[4] 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

[5] 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].

[6] 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].