Proceedings of the
2022 Improving
Scientific Software
Conference

Editors
Weiming Hu
Brian Vanderwende
Sheri Mickelson
Jesse Nusbaumer
The Technical Notes series provides an outlet for a variety of NCAR manuscripts that contribute in specialized ways to the body of scientific knowledge but that are not yet at a point of a formal journal, monograph or book publication. Reports in this series are issued by the NCAR scientific divisions, serviced by OpenSky and operated through the NCAR Library. Designation symbols for the series include:

**EDD – Engineering, Design, or Development Reports**
Equipment descriptions, test results, instrumentation, and operating and maintenance manuals.

**IA – Instructional Aids**
Instruction manuals, bibliographies, film supplements, and other research or instructional aids.

**PPR – Program Progress Reports**
Field program reports, interim and working reports, survey reports, and plans for experiments.

**PROC – Proceedings**
Documentation or symposia, colloquia, conferences, workshops, and lectures. (Distribution maybe limited to attendees).

**STR – Scientific and Technical Reports**
Data compilations, theoretical and numerical investigations, and experimental results.

The National Center for Atmospheric Research (NCAR) is operated by the nonprofit University Corporation for Atmospheric Research (UCAR) under the sponsorship of the National Science Foundation. Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

National Center for Atmospheric Research
P. O. Box 3000
Boulder, Colorado 80307-3000
Proceedings of the 2022 Improving Scientific Software Conference

Editors
Weiming Hu
Brian Vanderwende
Sheri Mickelson
Jesse Nusbaumer
Information about future workshops and other SEA news can be found on our website, --> [https://sea.ucar.edu/sea](https://sea.ucar.edu/sea)

The website of the 2022 conference is [https://sea.ucar.edu/conference/2022](https://sea.ucar.edu/conference/2022)

To be added to the workshop mailing list, please send an email to taysia@ucar.edu.
Proceedings of the 2022 Improving Scientific Software Conference

Table of Contents

Organizing Committee ........................................................................................................ iii

SEA 2022 Peer-Reviewed Papers ........................................................................................... 1-50

Toward designing effective exascale scientific computing workflows: experiences and best practices.................................................................................................................. 1-12
Mark Coletti, Russell B. Davidson and Ada Sedova

Multi-Container Actor Workflows For Supporting A Climate Science Gateway in Hawaii................................................................................................................................. 13-18
Michael Dodge II, Jared Mclean and Sean Cleveland

Marching Cubes Without Lookup Cases: Crawl the Edge Crossings Instead............................................................................................................................... 19-24
Carl Drews

Introducing XCast: A High-Performance Data Science Toolkit for Climate Forecasting.......................................................................................................................... 25-30
Kyle Joseph Chen Hall and Nachiketa Acharya

Continuous Integration of METplus Use Cases Using GitHub Actions and Docker.................................................................................................................. 31-34
George McCabe

Miranda Mundt, Wade Burgess and Dena Vigil

Communication Overlap In Large-Scale Spectral Transforms.................. 45-50
Dmitry Pekurovsky
Organizing Committee

Conference Chairs
Brian Vanderwende, National Center for Atmospheric Research (NCAR)
Sheri Mickelson, National Center for Atmospheric Research (NCAR)
Jesse Nusbaumer, National Center for Atmospheric Research (NCAR)

Program Committee Chairs
Weiming Hu, University of California, San Diego

Proceedings Committee
Weiming Hu, University of California, San Diego
Taysia Peterson, National Center for Atmospheric Research
Michael Flanagan, National Center for Atmospheric Research

Steering Committee
Brian Vanderwende, National Center for Atmospheric Research (NCAR)
Davide Del Vento, Quantinuum
Edward Hartnett, National Oceanic and Atmospheric Administration (NOAA)
Guido Cervone, The Pennsylvania State University
Jeffrey Carver, University of Alabama
Jesse Nusbaumer, National Center for Atmospheric Research (NCAR)
Joseph Schoonover, Fluid Numerics
Luis Metzger, National Center for Atmospheric Research (NCAR)
Maggie Sleziak, National Center for Atmospheric Research (NCAR)
Sheri Mickelson, National Center for Atmospheric Research (NCAR)
Shiquan Su, Intel Corporation
Srinath Vadlamani, Arm
Taysia Peterson, National Center for Atmospheric Research (NCAR)
Weiming Hu, University of California, San Diego

Workshop Administrator
Taysia Peterson, National Center for Atmospheric Research (NCAR)
Toward designing effective exascale scientific computing workflows: experiences and best practices

Mark Coletti
Russell B. Davidson
Ada Sedova
colettim@ornl.gov
davidsonrb@ornl.gov
sedovaaa@ornl.gov
Oak Ridge National Laboratory
Oak Ridge, Tennessee

ABSTRACT
Many fields within scientific computing have embraced advances in big-data analysis and machine learning as components of applications; these mixed approaches often require the deployment of large, distributed and complicated workflows that may combine training neural networks, performing simulations, running inference, and performing database queries and data analysis in asynchronous, parallel and pipelined execution frameworks. Such a shift has brought into focus the need for scalable, efficient workflow management solutions with reproducibility, error and provenance handling, traceability, and checkpoint-restart capabilities, among other needs. Here, we discuss challenges and best-practices for deploying exascale-generation computational science workflows on resources at the Oak Ridge Leadership Computing Facility (OLCF). We present our experiences with large-scale deployment of distributed workflows on the Summit supercomputer, including for bioinformatics and computational biophysics and deep learning model optimization. We also present problems and solutions created by working within a Python-centric software base on traditional High Performance Computing (HPC) systems; addressing such issues will be required before the convergence of HPC, Artificial Intelligence (AI), and data science can be fully realized. Our results point to a wealth of exciting new possibilities for harnessing this convergence to tackle new scientific challenges.

CCS CONCEPTS
- Software and its engineering → Software design engineering.

KEYWORDS
Scientific computing, workflows, high performance computing, software design

ACM Reference Format:

1 INTRODUCTION
Scientific campaigns that make use of neural network models together with traditional computational tasks have become more common. Training large, accurate neural network models requires substantial computational resources, as do many types of simulations such as those in particle and astrophysics and the molecular sciences. Therefore, emerging scientific computing efforts that make use of leadership-class HPC resources to implement these types of calculations have recently begun to incorporate complex, heterogeneous AI and simulation tasks that may run asynchronously on the resources, and may require multiple different tasks that each use the resources in different ways [6, 33, 54]. This is a shift away from the monolithic, synchronous, homogeneous computing patterns that have traditionally been deployed on leadership systems. The use of AI together with HPC simulation promises to provide increases in both accuracy and efficiency, and has been demonstrated for climate modeling [38], molecular simulation and materials science [30, 34], fluid dynamics modeling [10] and plasma simulations [3]. In addition to these types of hybrid simulation/AI workloads, some computational methods employ complex strategies to efficiently sample parameter spaces or simulated potential energy surfaces. Examples include optimization algorithms and hyperparameter tuning [13, 14, 40] and replica-based “enhanced sampling” schemes [9] in molecular dynamics simulations [11, 25, 28, 32, 36, 49, 51].

As leadership-class scientific computing enters exascale, these large heterogeneous workloads will require efficient workflow management schemes to enable both effective deployment and the ability to address fault tolerance, and error handling and provenance analysis when running tens of thousands of potentially asynchronous tasks of different types with complex dependencies on each other. Here we describe our experiences deploying several different examples of such workflows at scale on the pre-exascale Summit.
supercomputer at the Oak Ridge Leadership Computing Facility (OLCF), which can help to understand if efficient workflows on larger systems such as the Frontier exascale system at the OLCF can also be deployed. However, while our experiences are derived from work targeting leadership-scale deployments, our software approach, strategies, and lessons learned are applicable much more broadly, and will be useful for deployment of workflows on cloud resources and academic clusters.

2 BACKGROUND

The term “workflow” can have a number of different meanings from serial tasks in a particular order, to a set of jobs that may be performed on different resources. For business applications, workflows are generally seen as software that provides a formal way to describe a small set of repetitive tasks performed in a particular order [23]. Scientific workflows, in an analogous way, can be described as a defined set of tasks that must be completed in order for a scientist to go from a given set of data or a scientific problem to a scientific product. Scientific computing workflows will often involve many computationally intensive, heterogeneous tasks and are often data-oriented rather than control-flow oriented [5].

Here, in the context of HPC-based scientific computing workflows, we focus on large, parallel workloads that are deployed on the same parallel computing system, potentially having some components that may be performed in consecutive steps. Generally, these workflows will use a dataflow execution model, where compute resource units are given the next task in a queue when they become available. Any workloads that are complex or large enough will often require the use of a workflow manager, or top-level program that manages parallel distributed tasks and then reduces the data into a final form. For simple sets of completely parallel tasks, home-made scripts can be used. However, as the workflow becomes more complex these scripts create impractical amounts of work and potentially can back-up shared launch-nodes and file systems. Therefore, workflow management software created to explicitly manage task dispatch, work distribution, error analysis and fault tolerance, and other needs, was developed.

2.1 Workflow Management Software

In the realm of scientific computing workflows, a number of solutions have been developed, some of which are focused on more specific needs such as simulations for materials science or climate modeling [1, 18, 19, 21, 29, 50]. Some domain-specific tools for molecular simulations in biophysics aimed at increasing the sampling of the physical phase space have also emerged [8, 41].

Several of these solutions have used database management software as the underlying engine that manages connections between tasks. As far back as 1998, Ailamaki and co-workers noted that a database management system (DBMS) has many essential properties of a workflow management system and with a data-object view of scientific workflows the DBMS could be used to manage parallel computational tasks [4]. Since then, several scientific workflow management programs have incorporated DBMSs into their software design. For instance, Copernicus [42] uses the Python interface with sqlite, and Fireworks [29] and Radical Pilot [35, 48] use MongoDB for task management. Efficiency trade-offs stemming from the use of DBMS in scientific workflow programs have been discussed in recent literature within the context of HPC and massively parallel distributed solutions [46], noting that the analytical capabilities that the DBMS provides, while useful for analyzing workflow output and performance, can reduce efficiency of the workflow. Most recently, solutions based on pure Python with no third-party DBMS software have been deployed on HPC systems. The Parsl parallel scripting library enables the flexible creation of parallel workflows and has been shown to scale to 250000 workers across more than 8000 nodes [7]. The Dask parallel library also provides a scripting API for parallel task dispatch [43].

2.2 Scientific Workflows on HPC Systems

Because of the cross-fertilization between HPC simulation and the training of large neural network models, it is now being realized that a need for HPC and leadership-scale workflow managers exists to help deploy and manage potentially tens of thousands or more simulation, training and inference tasks. Some HPC-based workflow solutions focused on task dispatch rate for very large numbers of short tasks, and scaling to large numbers of tasks [7, 52]. The Swift/T program is such a solution, but requires the use of a special-purpose compiler to compile the workflow into the MPI framework. Reported task dispatch rate exceeded 60 thousand tasks/sec and Swift/T was able to use 64 thousand cores at greater than 90 % efficiency. However, the use of MPI reduces the ability of the workflow to be fault tolerant to the failure of workers [20].

Radical Pilot was deployed at scale on the Titan supercomputer at the OLCF for molecular dynamics simulations [36]. The MongoDB DBMS was tested for resiliency with respect to the connections required to launch over 15,000 tasks on the OLCF Summit supercomputer [37]. In later work, FireWorks was deployed at scale on OLCF Summit for structure-based drug discovery simulations for COVID-19 research [26]. While the resiliency tests succeeded on Summit, they required the launching of MongoDB on a compute node in order to access the ability to set the node ulimit t to a large value (~64K). Launching of FireWorks was performed through an external interface with another compute cluster known as the Slate Marbel system, a container orchestration system built on Kubernetes and OpenShift.1 In this case, ulimit t settings and connection failures caused some inefficiency and a more efficient dispatch was provided with an in-house workflow script using an in-memory DBMS. Dask has been used on Summit for launching large HPC workflows for evolutionary algorithms including the LEAP library [15].

Besides scalability, reduced overheads, and task dispatch rates, some other important requirements of large-scale HPC workflow software for managing simulation and AI tasks in emerging scientific efforts include ease of reuse, and the ability to create logging files and other workflow monitoring functionality that can efficiently and effectively be used on many thousands of asynchronous tasks. Here we describe our use of the Dask library to create HPC workflows that can satisfy these requirements, after briefly describing some of the different scientific areas that have recently found a need for such deployment solutions.

1https://docs.olcf.ornl.gov/services_and_applications/slate/overview.html
2.2.1 Workflow needs for evolutionary algorithms. Evolutionary algorithms (EAs) provide a scalable, efficient means for performing optimization and parameter exploration on parallel computing resources by not only exploiting the naturally parallelizable nature of EAs, but by doing so in an asynchronous manner [45]. On HPC systems (as opposed to cloud resources), a job allocation must be efficiently used without long periods of idle resources. Therefore, efficient deployment of EAs on HPC systems require some algorithmic modifications to prevent idle nodes. One such solution is the asynchronous steady-state evolutionary algorithm (ASEA), described in detail in §3.1.1 below. More complex combinations of different EA-driven neural network training schemes can require the workflow to enable communication between several different programs that each run on a large resource set.

One example of a complex HPC workflow between two ML components is shown in Fig. 1, and shows how gaps in training data can be filled in by using an adversarial EA to find those gaps. There, the EA Multi-node Evolutionary Neural Networks for Deep Learning (MENNDL) trains a deep learner (DL) model using the given training data; it will also perform hyperparameter and architecture optimization by evaluating thousands of different DL model configurations in parallel on Oak Ridge National Laboratory (ORNL)’s Summit supercomputer [39, 40, 53]. Then Gremlin takes the best model from MENNDL, which finds a set of features where the model performs poorly. This information is exploited to adjust the training data to add more examples corresponding to those poor performing feature sets. The cycle continues when MENNDL then resumes training with the updated training set.

The challenge here is that these two EAs can be run consecutively, but it may be more efficient to have them running simultaneously with a workflow tool intelligently managing their respective resources and ensuring that the two EAs work with one another effectively. Implementing such a complex HPC workflow on a system like Summit, and its successor Frontier, remains a challenge particularly because support for this type of dynamic workflow is beyond the reach of traditional bespoke batch scripts. That is, such scripts will not be able to dynamically adjust resource allocations (nodes) depending on the needs of the coincident EAs nor effectively communicate between them.

2.2.2 Workflow needs in computational biosciences. Biological molecular processes are some of the most complex to model, as they are highly non-linear, multiscale, and often modeled non-deterministically. Exponential growth in the number of known sequenced genomes, driven by decades of advances in gene sequencing technology, has created a widening technology gap: that of extracting evolutionary information and biological function from these sequences. In order to break through the data processing barrier in bioscience research, we require a multi-pronged computing strategy that can deal with heterogeneous problem sizes of computational molecular evolution, protein structure prediction, biomolecular simulation and structural analysis. Simulation tools, methods to analyze the flood of sequence and structure data, and the recent incorporation of AI have enabled advances in our ability to design and predict biomolecular systems for medicine, bioenergy and biosecurity. HPC can provide a way to help bridge some of the technology gaps caused by the data explosion. HPC workflows in biology can use a combination of tools such as deep learning based protein structure prediction, sequence alignment, and modeling/simulation, deployed at the genome scale [24]. An example of such a workflow we implemented on Summit is discussed in more detail in §3.1.2.

2.2.3 Workflow needs in computational chemistry and materials science. A recent explosion of data and AI driven methods in chemistry and materials science [2] has opened the door for the combination of HPC simulation, data science and machine learning [16, 54]. Recent work in this direction by our team has involved the deployment of programs for training neural network models of physical potentials for running fast molecular simulations with higher accuracy on larger systems. This creates a requirement for deploying complex workflows that launch training and simulation in parallel in order to create an optimized model that represents as much of the complete physical phase space as possible.

3 USING DASK TO SUPPORT HPC WORKFLOWS

In the following, we will describe some of the programming considerations, software design strategies, and optimizations we developed to enable the creation and deployment of workflows on HPC systems. For these workflows, we have focused on the use of the Dask parallel library, which is a popular python package for distributed processing that scales from laptops to supercomputers [17]. We discuss the steps taken to deploy these workflows at large scale on the Summit supercomputer, the work-arounds for problems we encountered, and the potential strategies for even larger deployments on upcoming exascale systems based on our experiences.
3.1 Two Types of Dask-supported HPC Scientific Workflows

Here we share two types of HPC workloads that we have implemented using the Dask parallel library, classified by software design pattern as dynamic or static in treatment of tasks. While some workflow management software tools may have higher level APIs for task dispatch, Dask programming allows for more control by the developer which can be useful when creating workflows on new systems at scale. The dynamic workflow allows tasks to be added during a run—e.g., for an evolutionary algorithm where new individuals have to be added to the task queue for evaluation, on the fly. The static workflow type entails processing a single, large batch of data for a fixed number of pre-defined tasks, and which was used to manage biomolecular workloads running structure prediction with AlphaFold2 [31] and molecular dynamics simulation with OpenMM [22] on Summit.

3.1.1 Dynamic workflows with Dask. Figure 2 depicts the workflow for an ASEA, which is an example of a workflow where new tasks are dynamically added while the workflow progresses. In this workflow, a single population of solutions, shown in the rounded box, is updated during a run. Individuals are evaluated on Summit nodes, represented by the GPU icons. When an individual is finished evaluation, it is compared with a randomly selected individual in the current population; the best of the two gets to stay in the population. The HPC resource used to evaluate that individual is now idle, so a random individual is selected from the population, cloned, and that cloned mutated to create a new, unevaluated offspring, which is then assigned to that resource to get it active again.

Listing 1: This Python code shows how Dask can be used to implement the ASEA workflow shown in Fig. 2.

```
init_pop = create_initial_pop()
futures = client.map(eval_ind, init_pop)
as_compl_iter = as_completed(futures)
for finished_task in as_compl_iter:
    result = finished_task.result()
    print(f'Evaluated individual: {result}')
    update_population(population, result)
    offspring = create_offspring(population)
    future = client.submit(eval_ind, offspring)
as_compl_iter.add(future)
```

The code in Listing 1 is an implementation for the ASEA workflow shown in Fig. 2. In lines 1–2 an initial random population is created and then distributed to all the Dask workers via the `client.map` call, where `eval_ind()` is a function that evaluates a single individual, and `init_pop` is a list of the initially generated random individuals. `client.map()` returns a list of futures that is used to get a Dask iterator in line 3; this iterator is used in the for loop in the next line to cycle through each individual as a Dask worker completes an evaluation. Normally a Dask user would call `client.gather()` to wait on all the futures, but we want to capture each individual as it is evaluated so that we can consider inserting it into the population, and then we want to create and assign a new offspring to its Dask worker so that it is busy again. Lines 5–6 updates the population with the new individual, either by replacing a current individual in the population with the new individual or discarding the new individual if its fitness is inferior. Line 8 a new individual, or offspring, is created, usually by randomly selecting an individual with replacement from the population, then that individual, or parent, is cloned, and then mutated, thereby creating an entirely new individual. This individual is then assigned to the previously idle Dask worker associated with `finished_task` so that it resumes work; this is done by calling `client.submit()` with the new offspring and the previously used `eval_ind()` function. However, of particular note is that the returned future is then used in line 11 to notify the iterator, `as_compl_iter`, of the new task. This is why we captured that iterator in line 3 so that we could later call its `add()` member function. (For those that use Python 3.8, or greater, you can use the new walrus operator to assign and use the iterator variable in the for loop declaration.)

This algorithm is better suited for HPC environments than the more traditional by-generation EAs. That is, progress to the next generation with the by-generation approach cannot proceed until all the new individuals of that generation have been evaluated; this is problematic if some individuals finish processing early, which means their corresponding HPC resources will idle until the next generation.
When a task was completed, the next protein was popped from the queue and assigned to it to begin processing. When a worker finishes processing, the next protein is popped from the queue and assigned to it to begin processing.

Figure 3: This diagram depicts a common workflow for processing biophysics data, in this case proteins, which are represented by the icons of strands of circles. The rounded box represents a queue of proteins awaiting evaluation, and the GPU icons correspond to Dask workers processing a protein. When a worker finishes processing, the next protein is popped from the queue and assigned to it to begin processing.

Listing 2: Example code demonstrating how to implement the workflow shown in Fig. 3.

```python
# Listing 2: Example code demonstrating how to implement the workflow shown in Fig. 3.

all_items = read_items()
futures = client.map(process_items, all_items)
for finished_task in as_completed(futures):
    result = finished_task.result()
    print(f"Finished processing: {result}")
```

Dask workers via client.submit() where process_items() will compute on each item in parallel. If the number of items exceeds the number of queues, then items not yet processed are held in the Dask scheduler’s queue. Items are popped from that queue and assigned to a Dask worker for processing as a worker completes a task.

We could have used client.gather() instead of the for loop, but the loop allows for tracking completed tasks. (Alternatively we could have implemented a Dask worker plugin that would monitor completed tasks\(^3\), but this approach is simpler and easier to understand.)

This workflow has also proven to be flexible enough to be readily applicable to other, similar problems. For example, it has successfully been re applied to allowing for massively parallel processing of molecular dynamics models using OpenMM, as described below in §4.1. We anticipate using this as a form of template for other problems that have similar computational needs.

3.2 Problem Solving and Optimizations for Dask Workflows

While scaling our Dask applications to work on Summit, we would occasionally encounter problems. Here, we describe some of those problems and their respective solutions or work-arounds. Later, we also discuss tuning Dask for optimal performance by changing configuration parameters.

3.2.1 Invoking Dask with class member functions. As seen earlier, submitting tasks to Dask for work involves invoking either client.submit() or client.map() with a single function and an item or items on which to apply that function per Dask worker. However, it may be the case that from time to time one will need to pass in a class member function to those functions, instead. This was the case for our EA that used the Library for Evolutionary Algorithms in Python (LEAP) toolkit because one requirement is implement the class that has a member function, evaluate() that is, in turn, directly called by Dask workers.

Listing 3 shows an example of a LEAP Problem class where a model must be run with a given set of features per Dask worker. That is, the task a Dask worker gets is to accept a set of features, here the phenome, and then calls the associated evaluate() with that argument. Models are generally large, so this version of the class tries to mitigate the load times by loading the model once in the constructor, and then just referring to the model in the function call.

However, though this is what normally happens when taking an object-oriented development approach, things are a little different.

---

\(^3\)https://distributed.dask.org/en/stable/plugins.html#worker-plugins
Listing 3: Example code showing a LEAP Problem class.

class ExampleLEAPProblem(ScalarProblem):
    def __init__(self, model_fpath: str):
        super().__init__()
        # Load the model here so we don’t need to reload for each evaluate(),
        # but this is actually bad.
        self.model = Model()
        self.model.load(model_fpath)

def evaluate(self, phenome) -> int:
    # Evaluate the phenome with the given model.
    :param phenome: is named tuple describing state
    :returns: score for model performance
    for this state
    ***
    fitness = self.model.predict(list(phenome))
    return fitness

Listing 4: Updated evaluate after use of client.scatter().

def evaluate(self, phenome) -> int:
    worker = get_worker()
    fitness = worker.model.predict(list(phenome))
    return fitness

Listing 5: Using run to have Dask workers load models at start of run.

def setup_worker(model_path=None):
    # setup worker to run model
    worker = get_worker()
    worker.model = Model()
    worker.model.load(model_path)

with Client() as client:
    client.run(setup_worker, model_path='some/model.pkl', wait=True)
    # ... regular Dask tasking follows

Another solution would be to use client.scatter(). That is, on the Dask client the model is created and loaded, and then client.scatter() is used to transmit that model to the workers so that it is stored locally. Then evaluate() can refer to the model stored in the worker, thus saving network and compute resources. The updated evaluate() that assumes the Dask client has done this is shown in Listing 4. Now the model is indeed loaded once in each worker, but as the scatter() documentation warns\(^4\), this is still not the ideal approach because the model still has to be transmitted to all the Dask workers, though admittedly this only occurs once.

---


---

Listing 3: Example code showing a LEAP Problem class.

Listing 4: Updated evaluate after use of client.scatter().

Listing 5: Using run to have Dask workers load models at start of run.
However, most of your attention will be on tuning parameters found in `~/.config/dask/distributed.yaml`. In that file most of the attention should be given to variables that control timeouts, heartbeats, and time-to-live (TTL). The timeouts are fairly self-descriptive, but the heartbeats bear some discussion. First, the client will periodically send a "heartbeat" to the Dask scheduler; if the scheduler has not heard from the client in a span of time, then it will assume the client is gone and unregister that client. This span of time is controlled by `distributed.client.heartbeat` and defaults to 5 seconds, and for Summit we increased that to 30 seconds. However, it is not the client heartbeat that will be the cause of the most potential woe, but rather a similar heartbeat for the Dask workers. Just as with the client if the scheduler does not get a heartbeat from a worker within a span, it will assume the worker is dead, and so will unregister it and reassign its task to another worker. The variable controlling this is `distributed.scheduler.heartbeat`, which is a little counterintuitive since naturally you would be looking for worker heartbeat, which does not exist. In any case, when running jobs with hundreds or even thousands of workers, the number of heartbeats the scheduler has to "listen" to can be overwhelming, so dialing up this variable may be very helpful, particularly if it is observed during sensitivity runs that workers are getting unregistered when they are actually still active. (However, if your Dask workers appear to be mysteriously dying, we suggest looking at this site for guidance: [https://distributed.dask.org/en/stable/killed.html](https://distributed.dask.org/en/stable/killed.html))

It may also take a while for connections to be made especially if a lot of workers, say thousands, are coming online. So other variables to address are `distributed.comm.timeouts.connect` and `distributed.comm.timeouts.tcp`, which both default to 30 seconds. For Summit, we have had to greatly increase those to 1800 seconds for large numbers of workers, but those values will differ for other systems, of course. We also recommend ensuring that worker stealing is on by setting `distributed.scheduler.work-stealing` to True; there is more discussion on worker stealing in the next section.

### 4 PERFORMANCE TESTING OF DASK WORKFLOWS AT SCALE

There are a number of parameters that can be tuned to enable Dask to both run successfully on large numbers of nodes, and to use these resources efficiently. An important functionality within Dask is the ability for workers to steal tasks from the pre-determined task lists that are created on initialization and assigned, and may be sub-optimal. With a dataflow execution model, if tasks have a variability in runtime, this can help significantly with reducing worker idle time and filling all available compute resources with work, especially if the initial task list creates an unbalanced assignment. Exact details about initial task assignments and work stealing policies are decided within settings connected with the scheduler, and may be difficult to tune and understand exactly. Some of the factors that Dask uses to determine how work stealing is performed is based on metrics that are used to determine runtimes of a particular function, and detectable locations of the data the workers will require to perform their task. Dask uses a moving average over function runtimes to obtain an approximation for how long a task is expected to run. If a particular function can have highly variable runtime based on input type, it is possible that this method will provide an inaccurate estimate. In addition, Dask tries to understand the communication requirements of distributed calculations and data locality, for treating partitioned matrix calculations and other distributed tasks with dependencies and data-sharing and communication requirements; for completely parallel tasks, these calculations may not be helpful for performance. In addition to the Dask decisions about task dispatch, it is possible that user-modifications to the list passed to Dask could also influence performance. To test these factors, we create a list of simple tasks with a large variation in time to solution, by specifically including a waiting step of a variable number of seconds drawn randomly from a distribution. We used this task list to query the final task order that Dask dispatched, and how much idle time resulted. We also tested the effects of turning the work stealing function on and off, and how a task list pre-sorted by runtime affects the Dask workflow that is dispatched.

Figure 4 shows the results of running the test workflow with work stealing turned on and on an input list of tasks that was sorted by runtime size, and one that was not. In many cases an approximate runtime can be predicted if a calculation scales as some function of the input’s characteristics. For protein structure prediction, for instance, runtime is dependent on sequence length. Figure 4 A shows the distribution that was drawn from to create the task runtimes for the test workload. B shows the distributions of idle times for each worker when using the sorted (green) and unsorted (blue) input list. Here, the sorted input list added to the overall efficiency by reducing the worker idle times. C and D show the worker tasks that were run for each set-up; the first set of tasks elucidates the initial task list that is created by Dask. In the case where work stealing is turned on and sorting is applied, it seems the initial task list distributes in a round-Robin manner, assigning a task from the list to each worker before returning to add a new task to the first worker’s list. Thus, the larger tasks are assigned to all workers first. Following this, some smaller tasks seem to have made their way in between two consecutive large tasks, and it would appear that some rearrangement of the task list must have occurred. For the unsorted input, the initial tasks appear to also be random, and for some workers during the workflow run, several consecutive large tasks were also assigned to workers.

Figure 5 shows the measured idle time for workers, for both the sorted and unsorted input lists, when work stealing is turned off. Behavior is very different this time, with the sorted input leading to substantially more idle time for the workers. An inspection of the two initial task assignments showed that in the sorted situation, tasks were not distributed in a round-Robin manner from the list, but rather the first, largest tasks were assigned to the first 45 workers. The last workers received only small tasks, and without the work stealing option turned on, no optimization of the queue was performed and many workers remained idle after completion of all of their tasks, while the first workers remained occupied with large jobs. On a system like Summit, this inefficiency is also expensive, in that there is no elasticity in the batch job scheduling system and a large job request will continue to use all requested nodes even when these nodes may be idle.

While these tests have demonstrated default Dask behavior, there are a number of parameters that can be set to control how the...
scheduler makes task lists, decides on work stealing procedures, and how work is prioritized when there are more tasks than workers. It is clear that performance and efficiency relies heavily on all of these setting and the decisions that Dask is making about how to distribute and execute work in the workflow. We see that in this case, it is more efficient to use the work stealing features even when tasks have no dependencies, no communication needs, and almost no data to be moved.

4.1 Deployment at Scale for Molecular Dynamics Simulations

Atomic-resolution structural models of biomolecules, such as proteins, nucleic acids, and membranes, can provide key insights into the system’s function and relevance in biochemical processes. Yet, these molecules are inherently dynamic with complex free energy surfaces that describe the system’s conformational states. Molecular dynamics (MD) is an efficient method to sample the conformational states of biomolecules by numerically propagating Newton’s equations of motion for atomic systems. The calculations performed during MD simulations are prime for GPU-acceleration and parallel computing methods on HPC resources. Furthermore, the Dask workflows presented here have been used to run massive amounts of MD simulations, making traditionally-expensive protocols like Replica Exchange MD, Adaptive Sampling, and other free energy calculations much more feasible.

To demonstrate a basic Dask workflow’s capability, a series of MD simulations are run as tasks. The OpenMM [22] simulation engine was used for running NVT simulations of the SARS-CoV-2 Mpro enzyme with a bound ligand and a solvent box of TIP3P water molecules [44]. The Amber ff19SB[47] and GAFF force fields were used as parameter sets for the protein and ligand, respectively. The Langevin dynamics thermostat was used to maintain a 310 K temperature. Particle mesh Ewald (PME) was used to approximate long range interactions with an explicit nonbonding interaction cutoff of 12 Å. The integration time step was 2 fs. Trajectory frames were written every 50,000 steps with a total of 250,000 steps performed (0.5 ns of simulation per task). Each MD simulation is given 1 GPU and 1 CPU core as resources.
The Dask pipeline was tested on 1 to 1000 Summit nodes, with 6 workers per node. The number of tasks were scaled by the number of workers, 3 per worker. The client waited for all workers to spin up before tasks began processing. Figure 6 shows the worker schedule of the MD workflow running on 45 nodes (270 workers). Red lines indicate the start and end of the Dask workflow while green bars indicate times during which a worker is running a MD simulation. The empty slots represent overhead times during startup of the Dask scheduler, Workers, and Client as well as the shutdown overhead as tasks finish and communications are stopped. Minimal overhead is seen between tasks as they are handed off to workers. Table 1 shows the average and standard deviation of these overhead times for the MD Dask workflow across a range of nodes. Generally, all overhead times increase as the number of nodes increases although this behavior is not linear. As more workers are provided to perform more tasks, the Dask Scheduler requires more CPU cores and communications between the Scheduler, Workers, and Client become more time consuming. For 1000 nodes, the dask-scheduler command was provided 20 cores rather than the 10 provided for the other runs of the workflow.

In terms of MD efficiency, the Mpro trajectories were obtaining $189.5 \pm 0.5 \text{ ns/day}^{-1}$. Since each task is performed independently, this simulation efficiency is not affected by scaling to large number of nodes. Therefore, each node can perform six trajectories of $\approx 190 \text{ ns in a day}$. Once scaled to hundreds or thousands of nodes, this Dask workflow can easily obtain hundreds of microseconds and up to milliseconds of MD simulation of biomolecular systems within a single day in the form of independent MD simulations, which overcomes potential pitfalls of an equivalently long single MD simulation. This amount of sampling is often required for experimentally-accurate quantitative analysis of a biomolecule’s conformational free energy landscape and/or the system’s kinetics [12, 27].

Table 1: Worker overhead times in seconds.

<table>
<thead>
<tr>
<th>Nodes (Workers)</th>
<th>Start (sec)</th>
<th>End (sec)</th>
<th>Between Task (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (6)</td>
<td>62.6109 ± 0.0003</td>
<td>8. ± 4</td>
<td>1.6458 ± 0.0002</td>
</tr>
<tr>
<td>2 (12)</td>
<td>64.2754 ± 0.0004</td>
<td>3. ± 4</td>
<td>0.6018 ± 0.0002</td>
</tr>
<tr>
<td>45 (270)</td>
<td>78.763 ± 0.008</td>
<td>8. ± 2</td>
<td>3.748 ± 0.008</td>
</tr>
<tr>
<td>500 (3000)</td>
<td>91.49 ± 0.09</td>
<td>100 ± 10</td>
<td>31.6 ± 0.2</td>
</tr>
<tr>
<td>1000 (6000)$^\dagger$</td>
<td>203.4 ± 0.2</td>
<td>30 ± 20</td>
<td>27.9 ± 0.5</td>
</tr>
</tbody>
</table>

$^\dagger$ CPUs provided to the Scheduler were increased to 20 cores to handle the increased communication overhead.

Code is available to recreate this Dask workflow at https://github.com/BSDExabio/OpenMM-on-Summit

5 DISCUSSION AND CONCLUSIONS

We have shown that the Dask parallel Python library can be efficiently used to launch large workflows on HPC systems both statically, where all tasks are defined completely at the launch of the workflow, and dynamically, where idle workers are assigned a
new task that is created on-the-fly based on the current state of the workflow. The latter is a strategy for optimization algorithm deployment such as in the use of asynchronous steady-state evolutionary algorithms.

We also shared Dask programming problems related to using class member functions when managing onerous common resources, such as using a model to make predictions within a Dask task. In addition, we discussed the necessity of tuning certain Dask configurations for timeouts, heartbeats, as well as specifying where Dask temporary files are written to address potential performance issues. Along the way we pointed the reader to supporting Dask documentation that may otherwise have been easily missed. We also pointed out Dask configuration variables on which to focus, that will need to be tailored for each HPC environment.

We found that enabling work stealing was important to performance when all other worker-task-assignment related parameters were used in their default settings, using benchmark tests with variable runtimes. Timeout settings and numbers of compute cores provided to the dask-scheduler helped prevent workflow failures for higher node counts. We demonstrated a large deployment of molecular dynamics simulations on OLCF Summit with 1000 nodes. While overhead times increased about 10 times when going from 45 to 1000 nodes, the total overhead is a small fraction of the total runtime for these simulations when used in production, and therefore is not a concern. Overall, it seems that using a Python parallel framework with no third-party database management software is an effective solution with a simpler installation for large-scale deployments on HPC systems.

In conclusion, we have found that reusable, scalable workflows can be deployed with a similar, lightweight Python framework for many different types of scientific computing efforts across many disciplines, and this approach can provide a strategy to manage the complex, heterogeneous workloads that are emerging as HPC simulation converges with machine learning.

ACKNOWLEDGMENTS

This research was sponsored by the Laboratory Directed Research and Development Program at Oak Ridge National Laboratory (ORNL), which is managed by UT-Battelle, LLC, for the U.S. Department of Energy (DOE) under Contract No. DE-AC05-00OR22725, and used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725. We thank Dr. Mu Gao of Georgia Institute of Technology for his help and support for this work.
Multi-Container Actor Workflows For Supporting A Climate Science Gateway in Hawaii

MICHAEL DODGE II, University of Hawaii - Information Technology Services - Cyberinfrastructure, USA
JARED MCLEAN, University of Hawaii - Information Technology Services - Cyberinfrastructure, USA
SEAN CLEVELAND, University of Hawaii - Information Technology Services - Cyberinfrastructure, USA

Fig. 1. Example Rainfall Mapping Workflow - 1.) Tapis Actor’s cron-scheduler deploys the container. 2.) The container executes the domain codes and utility scripts. 3.) The utility scripts use the Tapis Files service to transfer necessary metadata and monthly progress input files. 4.) The domain portion of the workflow executes to collect and process precipitation data. 5.) Workflow output files, metadata/annotations and execution logs are uploaded using the Tapis Files service to the HCDP.

The reproducibility of analysis is an important factor to the credibility of research and impacts the reuse of the research outputs and products. The rise in software container technologies has helped to address some issues around reproducibility, by providing a system where the tools necessary to rerun an analysis are packaged into a single, shareable machine image. How can we leverage containers in a reproducible and provenance tracked way to address workflows that must execute daily to support near-realtime analysis of climate data for the state of Hawaii? Also, how can we help support researcher development of the workflow and code evolution as new variables and data are added to the workflow? We present an approach that leverages the serverless actor API in the Tapis framework with its custom CRON-scheduling system, where parameters (as environment variables) of the container can be adjusted ahead of each run, and different tasks can be handled at arbitrary intervals as needed by researchers along with built in provenance of each execution tracked. Combining this with version control and a layered container structure to streamline the building and versioning of containers as new workflow data and products are added to the workflow code allows researchers to more easily push updates to the production workflow.

CCS Concepts: • Computer systems organization → Embedded systems; Redundancy; Robotics; • Networks → Network reliability.

1 INTRODUCTION

As climate change becomes more pressing, access to current information and data is important to researchers, decision makers, and the community in order to be able to prepare and respond. The University of Hawai‘i Information Technology Services Cyberinfrastructure team, the Hawai‘i Established Program to Stimulate Competitive Research (EPSCoR) Ike Wai (Water Knowledge) project, and the University of Hawai‘i Water Resource Research Center (WRCC) have spent the last several years developing gridded monthly and daily climate products that have not previously been available for the state of Hawaii (Figure 1). These gridded climate products are useful for water resource analysis and planning, ecological modeling, and other applications. Prior to these efforts, mapping products were produced at irregular intervals ranging from several months to several years, and maps for current timeframes were generally not available. By automating and streamlining data acquisition, quality assurance and quality control (QA/QC), gap filling, interpolation, and dissemination this initiative has been able to produce regularly
updated map products at the daily and monthly time scale and store them in the ‘Ike Wai gateway\[1\] making them accessible to other researcher and the community. With access to both historical and new near-real-time analyses, there can be facilitation of additional applications for these datasets, such as drought planning and wildfire management.

In order to support the broad scope of climate products that cross multiple science domains, an approach was adopted that allows separation of the various workflows and processes into self-contained executable software artifacts through the adoption of containers, such as Docker. This paper presents the organization and architecture of containerizing the domain codes used to produce the current near-real time products, how they are orchestrated to execute at daily and monthly intervals, and support for future products and workflows.

2 BACKGROUND

2.1 Tapis

Tapis\[6\] is an open source, NSF funded Application Program Interface (API) platform for distributed computation. It provides production-grade capabilities to enable researchers to 1) securely execute workflows that span geographically distributed providers, 2) store and retrieve streaming/sensor data for real-time and batch job processing with support for temporal and spatial indexes and queries, 3) leverage containerized codes to enable portability, and reduce the overall time-to-solution by utilizing data locality and other “smart scheduling” techniques, 4) improve repeatability and reproducibility of computations with history and provenance tracking built into the API, and 5) manage access to data and results through a fine-grained permissions model, so that digital assets can be securely shared with colleagues or the community at large. Researchers and applications are able to interact with Tapis by making authenticated HTTP requests to Tapis’s public endpoints. In response to requests, Tapis’s network of microservices interact with a vast array of physical resources on behalf of users including high performance and high throughput computing clusters, file servers and other storage systems, databases, bare metal, and virtual servers. Tapis aims to be the underlying cyberinfrastructure for a diverse set of research projects: from large scale science gateways built to serve entire communities, to smaller projects and individual labs wanting to automate one or more components of their process.

Tapis can be leveraged as a hosted solution or distributed between various institutions. Tapis is multi-tenant, meaning that there can be a number of organizations (i.e. a grouping of users, such as an institution, lab, or project) using the same set of Tapis API services but persisting data in logically separate, secure, namespaces. The central hosted instance is currently hosted by the Texas Advanced Computing Center (TACC) at the University of Texas at Austin. Other institutions, such as the University of Hawai`i (UH), have a hybrid deployment with subsets of Tapis API services deployed locally while leveraging others hosted at TACC. The UH Tapis instance is what the Ike Wai gateway leverages for backend API services.

2.1.1 Actors. Tapis leverages the Actor Model of concurrent computation and Docker, where users define computational primitives called actors with a Docker image and Tapis assigns each actor a unique URL over which it can receive messages. Users send actors messages via HTTP POST requests to the Tapis Actor API URL with the particular ID for the actor meant to receive the message. In response to an actor receiving a message, Tapis launches a container from the associated image and injects the message into the container. Most container executions are asynchronous from the message request, though Tapis does provide an endpoint for sending a message to an actor and blocking until execution completes. Tapis maintains a queue of messages for each actor, and is capable of launching containers in parallel for a given actor when the actor is registered as stateless. In addition to execution by message, Tapis allows Actors to be executed on a CRON schedule, a service for executing repeated tasks at a specified time, which is very useful in the case of daily execution for the climate workflows in this work.

To support the creation of these near real-time products the workflows need execution everyday, not only to generate near-real-time daily map products, but also to ensure the collection of transient climate data that becomes unavailable from some resource providers if not harvested within a window of availability.

2.2 The ‘Ike Wai Gateway

The ‘Ike Wai gateway supports research in hydrology and water management and provides tools to address questions of water sustainability in Hawai‘i. The gateway provides centralized web based user interfaces and APIs supporting multi-domain data management, computation, analysis, and visualization tools to support reproducible science, modeling, data discovery, and decision support for the ‘Ike Wai research team and wider Hawai‘i hydrology community. By leveraging the Tapis framework \[6\] UH has constructed a gateway that ties data management and advanced computing resources together to support diverse research domains including microbiology, geochemistry, geophysics, economics, humanities, and climate science. This platform hosts and disseminates the climate products through the Tapis APIs and allows the development of additional interfaces and integrations to support researchers.

3 WORKFLOW PROCESS CONTAINERIZATION

The various aspects of processing climate data and producing products ranging from precipitation\[3\] to air temperature\[2\] cross expertise and disciplines and the codes developed for these different workflows have typically been developed by different researchers or groups in a variety of programming languages. The adoption of a single monolithic code-base seemed unfeasible with the variety of dependencies and codes that can be introduced as additional data types and product types are added. Additionally, with the variety of groups working on development, the ability to develop across local and shared infrastructures and ease of production deployment made the use of software containers, like Docker, ideal for portability and consistency of execution environment across resources. The adoption of containers also allows for distributed execution of workflow processes over local or national resources such as High Performance Computing clusters or the Tapis Actor service cloud.
3.1 Container Hierarchy

During development of the container environments for including dependencies it was discovered that build times could take over 15 minutes when starting from a base operating system image such as Ubuntu 20.04. This build time overhead can be an impediment for frequent rebuilds or deployments that can take place during iterative development. In order to mitigate some of this overhead time the idea of a layered container hierarchy was adopted (Figure 2). The idea behind layering is to minimize repetition of installing libraries and scripts, and to make child or leaf layer builds as quick as possible. For example, all task containers are intended to work with Tapis, so Tapipy is installed as part of the task-base image. As a further example, data aggregation containers have extensive lists of dependency libraries that take a substantial amount of time to
Fig. 3. An example upload config file. Files and directories listed in the “upload” array will be copied to the corresponding remote path on the Ike Wai gateway. Permissions specified in the “dir_permissions” field will be applied to the remote directory. These files will be publicly accessible.

```
{
  "global_url": "https://",
  "notify": "by",
  "print_new_state": "true",
  "write_new_state": "false",
  "aged_options": {
    "set_server": "https://bucket.myserver.com",
    "connector": "remoteissor.myserver.com",
    "password": "remoteissor.myserver.com",
    "set_config": "remoteissor.myserver.com",
    "set_url": "remoteissor.myserver.com",
    "set_configs": "remoteissor.myserver.com"
  },
  "local_url": "https://bucket.myserver.com",
  "localギャモン": "remoteissor.myserver.com",
  "script": "remoteissor.myserver.com"
}
```

Fig. 4. Script Manifest of a data acquisition container.

download and build. The containers’ required libraries are added in at the appropriate layer’s build time. The container is named by concatenating the purposes of each layer. Accordingly, in Figure 3, the library-heavy Aggregation Base layer would be tagged as task-pre-agg. This way, whenever a new commit is pushed to the script repository, a new version of the container can be built in a few seconds. Additionally, utilities for uploading a workflow or processes products to the Ike Wai gateway repository are included in some of the parent layers so they are available to all the workflows. The upload utilities read a JSON file (Figure 3) that a researcher defines with the location of output products and what the destination should be on the Ike Wai gateway to make them available for dissemination and other workflow processes.

### 3.2 Enabling Transition To Containerized Workflows For Researchers

Climate science workflows can be rapidly expanded on and improved, which may involve frequent addition and removal of code and data. This project includes implementation utility scripts to parse simple human readable JSON definitions for a manifest to define what files should be pulled into the container and where they should be located in the file system within the container at build time. (Figure 4) The intent for this is to allow researchers to add or remove scripts and data files between builds without having to make changes to the container’s repository. Software library dependencies were identified and added to the Dockerfile for corresponding container layer in the hierarchy mentioned above. In order to make it easier for researchers that are developing the workflows to transition for their normal workflows of running shell commands to the containerized version of their codes there is a single shell script included in each container called task.sh which is set as the entrypoint for all container definitions. This allows the researchers to use the same commands they are currently using in their research workflows inside the container by simply placing them into that task.sh file. Also, to support portability across local script executions to containerized script executions, researchers can make use of environment variables to populate run-time parameters allowing the same parameter names to be used regardless of whether the container is executed locally or using the Tapis Actor API.

### 3.3 Workflow Development and Deployment

Development of the workflows has leveraged Jetstream[4] virtual machines (VMs) that have the Docker runtime environment installed. Researchers developed codes and workflows for generating monthly and daily rainfall and temperature products on the VMs and tested them using the local CRON service. The software code development was all tracked using git source code version control and GitHub as a code repository. As stated above software engineers worked with researchers to define manifest files and task scripts along with documenting dependencies for building the containers in the container code repository (https://github.com/ikewai/climate-automation).

Registering the workflows with the Tapis actors services leverages the Tapipy python library through a jupyter notebook template (Figure 5). With the manifest and upload JSON configuration files defined an image is built using a script that automatically tags the image with the current container repository commit hash id and the current climate code commit hash id in order to be able to derive the state of the source codes based on an image digest that can be tied back to the tag if debugging or provenance is being investigated. The image is published to DockerHub with the hashed commit tag as well as the “latest” tag. The Tapis Actor registration requires the definition of the image to be used, the environment variables that will be injected into the container at run-time as well as the CRON schedule to execute the container. With the Actor successfully registered the workflow process will begin execution at the defined time or schedule and the Tapis Actor API can be used to access the executions and logs.

### 4 FUTURE WORK

Automated container building and testing will be crucial to a full-fledged continuous integration and development pipeline, and further development is planned to use git-triggered webhooks with Abaco[7] to build, test, and update containers with minimal human intervention.
Fig. 5. Container Registration and Deployment
For improved tracking and debugging of produced containers, all of the contained domain and execution code stored in the container should have an associated commit hash. Furthermore, a manifest file can be generated and added to each container, logging its contents and sources to provide a simple method for reproducing the container’s build procedure.

Currently, Abaco/Docker containers are the only supported execution environment for workflow steps. Future workflows that require more dedicated computing resources can be set up to leverage the Tapis Jobs API for accessing on-site HPC or cloud computing resource.

Docker Hub is used to store built container images to be executed by the workflow. Adding support for additional container image repositories would increase the versatility of workflows and potentially reduce the reliance on third-party services.

The current implementation is set up to support workflows for rainfall and temperature data; however, this will need to be expanded to cover additional workflows for further climatological data, such as evapotranspiration and solar radiation. Additional workflows should be able to use the same general framework developed for this initial set with minor modifications based on the specific processing and computational needs of the data.

5 CONCLUSION

This paper has presented an approach to the organization and architecture of containerizing climate science domain codes to ease the transition of typical researcher workflow development into portable software artifacts that can be orchestrated across distributed cyber-infrastructure resources.

6 ACKNOWLEDGMENTS

This work was supported by the National Science Foundation grants - RII Track 1: ’Ike Wai Securing Hawai‘i’s Water Future NSF OIA #1557349 and the Tapis Framework: [1931439 and 1931575].

REFERENCES

Marching Cubes Without Lookup Cases: Crawl the Edge Crossings Instead

Carl Drews
drews@ucar.edu
Atmospheric Chemistry Observations & Modeling (ACOM)
National Center for Atmospheric Research (NCAR)
Boulder, Colorado, USA

Abstract
The Marching Cubes 33 algorithm [1] uses a lookup table of 33 cases to formulate a 3D isosurface of some scalar quantity. A cube in this context is 8 neighboring grid points in three-dimensional space. The author has developed an alternative algorithm that does not use a lookup table, but instead “crawls” around each cube along the edges of the isosurface by visiting adjacent points where the isosurface crosses an edge. The algorithm uses the central position of each edge crossing relative to the two adjacent vertices to resolve ambiguities in the configuration of the isosurface on each cube face; we expect to obtain a better isosurface by taking the crossing position into account. The paper presents diagrams showing how the isosurface is determined on each face, and is then extended to span the middle of the cube. The algorithm uses straight lines instead of the hyperbolae used in the asymptotic decider [6].

NCAR’s Atmospheric Chemistry Observations & Modeling laboratory (ACOM) seeks to improve 3D visualization techniques for atmospheric chemistry. We have implemented the “crawling” version of Marching Cubes [5] in Python and we are applying it to view output from chemical models. Our code derives isosurfaces of two atmospheric constituents: ozone and wildfire smoke. The National Oceanic & Atmospheric Administration’s (NOAA) chemical model WRF-Chem [7] calculates forecasts of chemical weather, and the isosurfaces are generated from model output on a daily basis. These isosurfaces are expressed in KMZ 3D model files; Google Earth provides a 4D viewing platform with a time slider to animate the shapes over time. The resulting visualization makes it easier for atmospheric scientists to analyze plumes of ozone or smoke as they evolve. The 4D visualization can inform where to construct more detailed cross-sections, and can guide flight planning for research flights.

WRF-Chem generates 48 hours of hourly output on a 12 km grid over the contiguous United States, with (390, 230, 43) grids points in the (x,y,z) directions. This output poses some challenges for generating and rendering 3D isosurfaces. The “cubes” near the surface are in fact thin rectangles only 20 m high, and the model coordinates follow the terrain and become tilted over mountain ranges. The paper explains how to solve these problems using cube coordinates for each model cell.

CCS Concepts:
• Applied computing → Earth and atmospheric sciences;
• Human-centered computing → Visualization design and evaluation methods.

Keywords: isosurface, marching cubes, 3D visualization, atmospheric chemistry

ACM Reference Format:

1 Introduction
Research in atmospheric chemistry seeks to understand and forecast the behavior of chemical compounds and aerosols (suspended particles) in the earth’s atmosphere. Of particular interest at NCAR’s Atmospheric Chemistry Observations & Modeling laboratory (ACOM) are smoke plumes from wildfires and ozone generated by industrial processes over cities. Figure 1 shows smoke plumes emitted by the River Complex Fire in northern California on August 19, 2021. Figure 2 shows an “ozone bubble” that formed over Kansas City on June 10, 2021. The chemical concentration typically decreases with distance away from the source of emissions. Isosurfaces of constant concentration (often measured in parts per billion) provide a way to visualize the plume or cloud of chemicals in three dimensions. Nested isosurfaces can reveal the progression from higher to lower concentration as the chemical mixes with clear air away from the source of emissions.

Atmospheric models commonly represent the atmosphere in a gridded lattice of rectangular cells located above the
Figure 1. Fire locations and smoke emitted from the River Complex Fire in Northern California on August 19, 2021. Note that the two major smoke plumes are blowing in different directions at different altitudes.

Figure 2. WRF-Chem simulated ozone over Kansas City on June 10, 2021. The prevailing winds have blown the urban ozone toward the south. The inner red isosurface represents ozone concentrations greater than 75 parts per billion by volume, and the yellow isosurface represents ozone greater than 70 ppbv.

Figure 3. The 15 triangulations of Marching Cubes by Lorensen and Cline in 1987. This paper uses the orange dots to denote cube vertices that are less than the isovalue and therefore outside the isosurface. Image by Jmtrivial at Wikimedia Commons, GNU General Public License. (https://commons.wikimedia.org/wiki/File:MarchingCubes.svg).

When the author encountered the Marching Cubes algorithm in 2020, the 8-year span between 15 lookup cases and 33 cases suggested that this algorithm would be difficult to implement and test, since an error had gone undetected for several years. Lewiner in 2003 implemented 730 sub-cases in their enhanced lookup table [4]. Furthermore, I was concerned about the combinatorial explosion when going from a square with 4 vertices and $2^4 = 16$ cases to a cube with 256 lookup cases, and the much greater number of lookup cases that would result if we ever wanted to extend the isosurface in the time dimension as well. This paper describes an alternate implementation: launch a process to explore the cube and traverse the edges of the inner surfaces shown in Figure 3 instead.
2 Edge Crossings

Nielson and Hamann describe bilinear interpolation within the faces of a marching cube [6]. Interpolation along the edge of one face calculates the point at which the isosurface crosses that edge. For example, if one end of an edge has a value of 5, and the other end has a value of 13, and the isosurface value is 10; then the isosurface crosses that edge at a distance of \((10-5)/(13-5) = 0.625\) from the first end (the face is a normalized square with side length = 1.0). Each face can have 0, 2, or 4 edge crossings. Consider the three example triangulations shown in Figure 4. Our goal is to collect the \([x,y,z]\) points around the perimeter of the orange interior surfaces.

2.1 Crawling Around the Cube

Figure 4. Three example triangulations to be explored by crawling the edge crossings. The orange dots indicate cube vertices outside (below) the isosurface. Derived from image by Jmtrivial at Wikimedia Commons, GNU General Public License. (https://commons.wikimedia.org/wiki/File:MarchingCubes.svg).

In Figure 4a the goal of the algorithm will be met by crawling around the front lower-left corner of the cube, thereby generating a single triangle that represents the isosurface at three edge crossings. In Figure 4b the isosurface cuts through the middle of the cube; this case is satisfied by crawling around the waist of the cube and returning 4 points in \([x,y,z]\) space. Figure 4c shows two separate crawls; the lower front two corners describe a rectangular crawl, and then the algorithm must execute a second crawl to capture the triangle around the upper right back vertex.

The algorithm populates a data structure for each cube encountered. The Cube class in Python is a 3x3x3 matrix containing:

- 8 Corners
- 7 Faces, including one unused face in the center
- 12 EdgeCrossings

The 7th unused “face” in the center of the cube is a dummy placeholder in the 3x3x3 matrix, for a total of 27 elements. The EdgeCrossing class has a boolean field to indicate if it has been visited, and a list to keep track of 0 or 2 neighboring edge crossings on adjacent faces. The crawling algorithm proceeds in three major phases:

1. Calculate all the edge crossings.
2. Resolve the contour lines on each face by connecting their edge crossings.
3. Crawl the connected edge contours to generate polygons.

The edge crossings are pre-calculated by linear interpolation between the end vertices. If a cube face has 2 edge crossings, they are connected as neighbors. If a cube face has 4 edge crossings, then we must resolve the face ambiguity according to rules in the following Section 3. When the edge crossings are properly connected, that 2D face becomes a valid contour plot with connected contour lines across the face. Finally the crawler traverses all the connected edge crossing, recording their \([x,y,z]\) coordinates while traveling, and generates a set of polygons in 3D space.

2.2 Collect Triangles and Higher-Order Polygons

The three \([x,y,z]\) points of a triangle define a plane. Any four points in the 3D space of atmospheric chemistry are almost certainly not co-planar, but form a twisted rectangle, or saddle, instead. Figure 4b shows a rectangular surface divided along its center diagonal into two triangles. This triangulation does not capture the true nature of a saddle, where the center point has an altitude midway between the surrounding four corner points. With this in mind, for rectangles and higher-order polygons we have chosen to introduce a center vertex that is the \(x,y,z\) average of all the perimeter points (Figure 5). The algorithm generates triangles on the isosurface by rotating around this center point. Rectangles are divided into four triangles instead of two. Nielson and Hamann also took this approach [6]; it removes the possibility of any triangle of the isosurface lying on the face of a cube. Figure 6 shows how the center vertex produces twisted rectangles of ozone in a bubble over Atlanta, Georgia.

Figure 5. When triangulating rectangles and higher-order polygons, we introduce a center vertex and rotate around it to generate 3D triangles.
Google Earth and other rendering engines require a normal vector for each triangle; this normal vector is used to calculate the shading with respect to the scene’s light source, and to assign a darker hue to the inside of the isosurface. ACOM’s algorithm for Marching Cubes first determines the outside direction of the surface by examining the values at the cube vertices, then encodes the sign of the normal vector into each triangle itself according to the right-hand rule. Consider the polygon in Figure 5. We know that this polygon has normal vectors pointing up out of the page, because the triangles are ordered rotating counter-clockwise around the center vertex. The face we see here is the outer side of the isosurface and will be rendered lighter than the underside. If the normal vector were determined to point down instead, then the Python code would reverse the list of vertices before delivering the triangles to Google Earth. Figure 7 shows a cut-away view of ozone over Denver that reveals darker surfaces inside the bubble.

Figure 7. Cut-away view of ozone over Denver, Colorado, on June 1, 2021. Cutoff height is nominally 3000 meters above sea level.

3 Face Ambiguity

Marching Cubes encounters a face ambiguity when any single face of a cube has four edge crossings and thereby forms a topological saddle (Figure 4c right face). Two opposite corners are higher than the isovalue and the remaining two corners are below (Figure 8). The color of the center of the cube face cannot be determined from the sign (+/-) of the four corners alone. Does Aspen lie on a high ridge connecting Cortez with Julesburg (red), or within a low valley stretching between Dinosaur and Springfield (green)? In Figure 8 there is no numerical way to resolve the center contour color at Aspen because the isosurface at 10 falls exactly halfway between the corner values of 5 and 15.

Figure 8. Face ambiguity illustrated in a 2D contour plot, with four edge crossings at contour value = 10, using a hypothetical map of Colorado by example (WRF-Chem grid cells at 12 km are much smaller). The color red represents terrain above the contour value of 10, and green designates geographical areas below the contour value. The gray diamond in the center represents an area whose height has not yet been determined.

The saddle ambiguity requires more information to resolve without relying on arbitrary tie-breaker rules (closest to origin, right-hand option first, etc.). We may resolve the ambiguity by treating the face as a 2D contour plot. What would this plot look like if we were simply trying to color in the square properly? The contour plot contains additional information in the calculated position of the edge crossings along the edges; in atmospheric chemistry those crossing locations will almost never be located exactly at the midpoint. The floating-point locations provide a guide for assigning the center color of Figure 8.

Since the edge crossing positions are calculated by linear interpolation along the edges, the diagonal contour lines across the center of the square represent the remaining uncertainty in the final contour plot. We want to minimize uncertainty and allocate interior regions in accordance with the known edges. The ACOM algorithm resolves the face ambiguity by connecting the edge crossings that result in the shortest sum of the two diagonal lines across the face.
We execute these calculations and assignments for each grid cell during phase 2. Aspen lies on a broad ridge between Cortez and Julesburg (9b), not within a narrow canyon connecting Dinosaur with Springfield (9a).

Our strategy produces results similar to the Asymptotic Decider [6]. Since the face ambiguity is resolved from information contained on that face alone, we can be certain that the interior isosurfaces of the adjacent cubes will match at the face and not leave any holes in the overall isosurface.

**Figure 9.** The face ambiguity is resolved in favor of the configuration that results in the shortest length of diagonal lines across the 2D cube face (Panel b). These diagonal lines are contours along the isovalue of 10; in phase 3 each contour line here will form one edge of an isosurface triangle projecting into 3D space.

### 4 Three Coordinate Systems

Running the ACOM Marching Cubes algorithm on 48 hours of WRF-Chem model output will often produce tens of thousands of triangles, depending on where the numerical thresholds are set for the green, yellow, and red isosurfaces. Within the Google Earth KMZ archive we organize the surfaces into folders by chemical (O3 or CO from fires), 5 altitudes (surface, troposphere, stratosphere, mesosphere, thermosphere) and 3 value ranges (green, yellow, red). Google Earth (GE) provides a time slider to step through the hourly output or run an animation. We have found it best to keep the size of the KMZ archives below 10 MB for best performance on a desktop. Since altitude is important and difficult to determine with the GE viewer, we provide a partially transparent horizontal overlay that can be adjusted up and down to measure the height of plumes and clouds.

Google Earth uses the concept of a “3D model” as part of its 3D rendering process. Models are encapsulated in COLLADA files [3]. We build one COLLADA file for each combination of hour, threshold, and altitude; that 3D model contains all the triangles for that isosurface even when there are discontinuous clouds (Figures 2 and 7 red). In theory a 48-hour WRF-Chem run could generate \(48 \times 5 \times 3 = 720\) COLLADA files; in practice many of those combinations do not exceed the minimum chemical threshold and are skipped. A typical KMZ contains about 100 COLLADA files with extension .dae. 3D models in Google Earth are placed at a single coordinate of \([\text{latitude}, \text{longitude}, \text{altitude}]\). The internal coordinates of a 3D model are in kilometers and are strictly Cartesian, without any correction for the curvature of the earth. For regions as large as the state of Colorado, this curvature is significant; straight lines tangent to the center of the state will “fly off into space” at the state borders. Ozone and wildfire smoke interact with the earth’s terrain. Consequently, we use a Cartesian origin at the lat-lon center of our model domain, and bend the model coordinates downward and poleward to account for the curvature of the earth and its lines of latitude. For a straight line tangent at the middle of the Colorado - New Mexico border, this correction at Four Corners is calculated to be 5.7 km north and 7.6 km down. To summarize, the three coordinate systems involved in the ACOM Marching Cubes algorithm are:

1. WRF-Chem latitude, longitude, and height above terrain in meters.
2. Normalized cube coordinates for each grid cell.
3. COLLADA coordinates in kilometers, adjusted for the spherical earth.

### 5 Conclusion

The ACOM implementation of Marching Cubes uses a “crawling” procedure to replace the lookup tables of prior implementations, replacing what may be an error-prone test process for many cases and sub-cases. Our crawling process also avoids the combinatorial explosion that would result if a fourth dimension (time) were added to formulate a future Marching Hypercubes algorithm \((2^{16} = 65,536\) lookup cases). In addition, we have implemented two enhancements described by Nielson and Hamann: introducing a central vertex for rectangles and higher-order polygons, and using the accurately calculated location of each edge crossing [6].

What is the purpose behind 3D visualization of model results? A three-dimensional view introduces several serious difficulties into the art of data display: obscuring, axes, perspective, geo-coordinates, and limited number of colors for scaling (here 3 vs. a dozen or more on a typical flat contour map). The extra dimension triggers a combinatorial explosion in the amount of data to be processed, especially when the time dimension is included, and this explosion can overwhelm 3D viewing systems. An earlier ACOM prototype that used cubes instead of an isosurface caused serious performance problems for the Google Earth viewer.

For the May 28, 2022 WRF-Chem model run, the ACOM algorithm extracted ozone in 2 minutes and 34 seconds on one of our lab computers. The code examined 43,200 cells containing some portion of the isosurface, generated 151,414 triangles, and encapsulated them into 90 3D models in Google Earth (a placemark stored in one .dae file). The KMZ archive is 5.6 MB, which is quite manageable to view with Google Earth. The Google Earth viewer appears to be optimized to
favor individual model complexity over the number of 3D models; the same extraction by the earlier cube prototype would have produced over 43,200 3D models and would have rendered the scene too slowly to be practical. The isosurface approach keeps interactive 3D visualization within the performance range of desktop computing.

Consider the time-based curtain plot shown in Figure 10. ACOM made this curtain plot to compare WRF-Chem with actual measurements of ozone taken by the LMOL lidar site at NASA Langley in Hampton, Virginia during February 5–6, 2022. The plot provides a finely detailed color scale that shows the ozone concentration above LMOL with height and time. One can read the chemical values with accuracy. But beyond those advantages, it is difficult to determine what was happening here? Where did the sudden influx of ozone come from? From a particular compass direction, or is this a stratospheric intrusion from above? Why does the plot show a long lingering ”tail” that persisted at 2 km for 36 hours? And does that tail have any relation with the return of high-altitude ozone activity at 8 km at time 36 hours?

These questions are usually addressed by viewing a series of section plots; sequenced by time, latitude, longitude, and altitude. A surrounding grid of time-based vertical profiles would also help to determine the overall scenario. The author has implemented a click-plotting map that provides quick access to neighboring section plots. And yet, these are awkward ways to comprehend the 4D evolution of a synoptic weather system over time.

3D and 4D visualizations of the geosciences can be properly viewed as companion products to flat maps, curtain plots, and section plots. A firm understanding of the 4D movement will direct where and when to locate the 2D contour plots for maximum insight. Recent technical advances in Graphical Processing Units (GPUs), processor speeds, and Internet connectivity have brought 4D visualization within reach of standard office equipment in a scientific laboratory. The author directs readers to the GitHub repository of Python code for this project [2], and welcomes comments for improvements to ACOM’s algorithm and 4D visualization system.

Acknowledgments
The author is grateful to Mijeong Park, Wenfu Tang, and one anonymous reviewer for their helpful comments and suggestions.

References
Introducing XCast
A High-Performance Data Science Toolkit for Climate Forecasting

Kyle Joseph Chen Hall†
International Research Institute for Climate & Society
Columbia University Climate School
Palisades, New York, USA
hallkjc01@gmail.com

Nachiketa Acharya
CIRES, University of Colorado Boulder
NOAA/Physical Sciences Laboratory
Boulder, Colorado, USA
dr.nachiketaacharya@gmail.com

ABSTRACT
Climate prediction research products and operational climate forecasts are often made by using machine learning models to find relationships between gridded General Circulation Model (GCM) outputs and observed climate data, one gridpoint at a time, independently. These gridpoint-wise operations generally fall into one of several categories: statistical post-processing, GCM bias correction, or multi-model ensemble forecasting. No matter the approach, the implementation of gridpoint-wise operations serves as a significant barrier to entry to climate data science.

While there are powerful utilities for manipulating gridded data in Python, and numerous libraries and toolkits for two-dimensional statistical modeling and machine learning, extremely few support this gridpoint-wise approach to climate prediction. The need to bridge the gap between Python’s data science utilities and its gridded data utilities is evident. Additionally, Gridpoint-wise operations, like those used in multi-model ensemble forecasting, are time consuming and computationally expensive. Their spatial independence theoretically allows them to be parallelized, but in practice, implementing parallel computation in Python is a specialized skill which requires a large time investment. If climate scientists must also become computer scientists to implement their experiments, they have less time to spend on domain-specific research. While there are many domain specific tools for analyzing climate data and making forecasts like the NCAR Command Language, the Climate Predictability Tool, and Google Earth Engine, they lack the flexibility and accessibility required to be integrated easily with Xarray-based scientific workflows in Python. The inaccessibility of modern High-Performance Computing (HPC) tools therefore presents a significant barrier to entry to climate data science.

In order to address both problems, we present a flexible, easy-to-learn, and highly-performant set of tools for geospatial data science called XCast. XCast abstracts out the use of modern HPC tools, making chunk wise parallelism and cluster computing available to the user through an API which closely mimics those of traditional Python data science utilities. It allows Python data science tools like Random Forest or Pruned ELM to be efficiently applied to gridded data with minimal time investment, and implements climate forecasting tools like cross-validation and forecast verification modules, based on WMO recommendations. Here we describe the co-development of XCast and demonstrate XCast’s flexibility, extensibility, and high-performance nature through a case study of multi-model ensemble climate forecasting.

CCS CONCEPTS
• Applied computing ~ Physical sciences and engineering ~ Earth and atmospheric sciences
• Computing methodologies ~ Machine learning
• Computing methodologies ~ Parallel computing methodologies ~ Parallel algorithms ~ Massively parallel algorithms

KEYWORDS
Geospatial Data Science, Machine Learning, Parallel Computing, Global Climate Models

ACM Reference format:

1 Introduction
In climate prediction, learning relationships between gridded climate model outputs and observations is a core research technique. There are a number of ways to approach the application of statistical and machine learning techniques, like pattern recognition and aggregation, but it can be desirable to apply a given technique independently at each point in space. When spatial relationships are accounted for during the generation of a gridded dataset, as is the case with General Circulation Model (GCM) bias correction and climate forecasting, it is useful to examine the predictability of the observed variable independently at each point. (Acharya et. al., 2014) Gridpoint-wise operations such as these, however, present a significant barrier to entry for the earth scientist. For those without specialized training in computing, the manipulation of gridded...
multidimensional data and the pointwise application of statistical and machine learning-based methods can both be difficult tasks. The efficient application of gridpoint-wise operations and the subsequent analysis of results composes a logistical challenge which can take up undue amounts of an earth scientist’s time. Inefficient solutions to this logistical challenge can even exacerbate the already-significant computational overhead associated with gridpoint-wise operations.

Fortunately, there are ongoing efforts by the Python Earth Data Open Source community (Pangeo) to facilitate these processes. The manipulation of gridded multidimensional datasets in Python is handled by Xarray, a powerful library supported by a large open-source community. (Hoyer, S., Hamman, J., 2017) Xarray data structures have been largely adopted as the de facto atomic units of gridded data in Python, enough so that an ecosystem of climate data analytics libraries designed to produce and consume them has emerged. This ecosystem also includes climate data analytics libraries designed for dynamical modeling like MetPy, (May et al., 2022) ensemble production and forecast analysis packages like ClimPred (Brady, R., & Spring, A., 2021), tools for generating climate indices like XClim (Logan & Travis et. al., 2022), distributed model training tools like Dask-ML (Rocklin, 2015), and many others. Python also implements a diverse set of statistical and machine learning libraries, designed for the production and consumption of traditional flat datasets. Scikit-learn, a commonly used machine learning library, is one of many such Python data science utilities that operates nearly exclusively on two-dimensional datasets. (Pedregosa et al., 2011) Like scikit-learn, Python’s data science utilities usually operate on two-dimensional data, and are not designed to accommodate gridpoint-wise operations on Xarray data structures. This gap between Python’s flat data science utilities and its gridded data ecosystem is notable. XCast, a high-performance data science library for climate forecasting, designed by the authors, serves to bridge that gap by facilitating the use of Python machine learning and statistical toolkits with gridded data.

2 Development

XCast derives its name from the fundamental data structure it consumes, the Xarray Data Array, and its initial purpose, climate forecasting. It grew out of PyELM-MME, a platform for Extreme Learning Machine (ELM)-based Multi-Model Ensemble (MME) forecasting. (Acharya & Hall, 2021) The PyELM-MME platform was intended to make ELM-based Multi-Model Ensemble climate forecasting accessible. As the platform grew, the need for a generalized version supporting a broad array of data science tools became apparent. PyMME, the successor to PyELM-MME, emerged to fill that gap. (Acharya & Hall, 2021) However, during the PyMME development process, a major pain point became apparent: the gridpoint-wise operations implemented for multi-model ensemble forecasting could not be used outside of the PyMME forecasting setting. Since both PyMME and PyELM-MME were implemented specifically in Jupyter Notebook environments, they were not easily usable in other cases.

The need for a generalized gridpoint-wise data science library was evident. We realized that, since the gridpoint-wise approach to data science operations is so broadly applicable in the earth sciences, any MME climate forecasting platform should be built as a special case of a more general gridpoint-wise data science toolkit. There is no reason to require earth scientists in other domains to retrofit an MME forecasting system to suit their own needs, when a library can be designed to make all gridpoint-wise data science operations accessible at once. Abandoning the Jupyter Notebook-based forecasting platform design, we implemented XCast as a general-purpose Python library so it could be used in any given environment, and decided to distribute it with Anaconda in order to make the installation process fast, easy, and compatible with other similar libraries. The first stable version of XCast, version 0.5.0, was released in March, 2022, and is available on Anaconda for installation on any operating system.

3 Software Design Principles

XCast’s implementation pursues three core principles: ease of use, flexibility, and high performance. These principles serve to support the goals of making gridpoint-wise data science operations as accessible as possible, and lowering the barriers to entry to computational climate science. Difficult installations and steep learning curves would definitionally present large barriers to entry to the user and, while less overt, a library’s lack of flexibility also serves as a barrier to entry to earth science. Additionally, needing to learn multiple programming interfaces, rather than a single one, increases the time investment required to accomplish the same task. The computationally intensive nature of gridpoint-wise operations can also serve as a barrier to entry to earth science since it increases the time investment even further. A high-performance library requires a smaller investment of time and resources to accomplish the same task. XCast attempts to accomplish all of this in one package.

3.1 Ease of Use

In order to minimize the time required to learn how to use XCast, its application programming interface (API) was specifically designed to leverage users’ prior knowledge. Xarray Data Arrays were chosen as the atomic unit of XCast’s analyses because of the high degree of Xarray adoption by the Python earth science community. Numerous other Python climate analytics libraries also adopt Xarray Data Arrays, which makes those libraries inherently compatible with XCast. Any user of those libraries, or of Xarray itself, would already have the requisite skills to use gridded data with XCast. Additionally, XCast’s API is intentionally designed to mimic that of scikit-learn, perhaps the most popular and well-supported Python data science library. The similarity between XCast’s API and scikit-learn’s API facilitates the transfer of skills and knowledge from traditional Python data science work to gridpoint-wise data science work. The treatment of two-dimensional NumPy arrays in scikit-learn is very similar to the treatment of four-dimensional Xarray DataArrays in XCast, so...
Introducing XCast

skilled scikit-learn users can apply their existing skills to XCast workflows. Leveraging the prior knowledge of the target user base makes XCast easy to learn, while its distribution venue, Anaconda, makes XCast’s installation process familiar to any user of Xarray or its supporting libraries.

3.2 High Performance

In pursuit of high-performance, XCast uses Dask to implement chunk-wise parallelism. Gridpoint-wise operations are uniquely parallelizable, since the computations at each point are completely independent of one another. The parallelism in XCast significantly decreases the time investment for gridpoint-wise data science operations, and XCast’s natural compatibility with Dask allows it to be scaled up to institutional computer clusters with ease.

The gridpoint-wise approach to multidimensional gridded data operations exemplifies the benefits of parallel processing. Theoretically, each grid point’s computation could be performed by a separate computer. Although unrealistic, that level of parallelism would effectively eliminate the spatial dimensions of the dataset, and reduce the time required to complete all the gridpoint-wise operations to the time required by a single gridpoint. Fortunately, it is possible to implement a more generalized version of the concept: chunk-wise parallelism. (Dab & Slama, 2017)

Rather than parallelizing gridpoint-wise operations by splitting individual grid points, this type of problem can be parallelized by splitting a dataset into spatially local groups of grid points called chunks. Each chunk can then be distributed to an independent processor, which reduces the time required proportionally to the number of chunks. In fact, the parallelizing operations by splitting individual grid points can be understood as chunk-wise parallelism, with a chunk size of one.

In cases where only a small number of processors are available to implement chunk-wise parallelism, the performance gain or reduction in computation time is not strictly proportional to the number of chunks. When there are more chunks than processors available, each processor must perform computations for several chunks in serial- losing time, compared to perfect parallelism. Practically, this often still represents a significant performance boost when compared to fully serial operations. Even further, since XCast implements chunk-wise parallelism by leveraging Dask, a high-performance Python multiprocessing library, XCast can easily be scaled to institutional supercomputers and computer clusters. Using Dask to scale XCast’s gridpoint-wise operations to powerful machines lets the user bypass Python’s global interpreter lock and further approach the perfect case of parallelism. Additionally, XCast gives explicit control over the level of parallelization to the user, by allowing them to specify the size and number of chunks so they can optimize XCast’s performance based on the specifications of their work station.

In an effort to identify cases where XCast outperforms other methods, and to find optimal chunking schemes and cluster configurations, here we analyze six different implementations of gridpoint-wise operations. The first method implements a naive for-loop implementation, which iterates over the spatial dimensions in serial. A native Xarray function called “apply_ufunc,” which applies arbitrary functions along predefined core data dimensions, was implemented for comparison. Four configurations of XCast were also implemented, for comparison. All tests shown were performed on a Mid-2015 quad-core MacBook Pro with Intel i7 processors.

Figure 1: XCast performance compared to for-loop and Xarray implementations of gridpoint-wise operations.

Figure (1) compares how the six different implementations of gridpoint-wise operations scale with intensity. The slowest method is the for-loop implementation, which is intuitive, because it forgoes any attempt at chunk-wise or point-wise parallelism. Notably, when XCast is configured with one chunk, and uses a single processor, its performance is nearly the same as Xarray.apply_ufunc. This can be explained by the fact that XCast and Xarray use Dask to implement their array operations.

One conclusion that can be drawn from figure (1) is that when a local Dask cluster is dispatched, or the dataset is split into multiple chunks, overhead is introduced which decreases performance. While each of the implementations except the for-loop method use Dask, the treatment of the underlying data and computation varies greatly from method to method. The single-chunk, single-core XCast method, represented in green in figure (1), treats the dataset as one contiguous chunk. While it is technically implemented with Dask, this behavior essentially reduces to that of an in-memory NumPy array. It also uses only a single process to perform gridpoint-wise operations in serial. For the multi-chunk, single-process XCast implementation, represented in red, the dataset is split into 30 chunks, and treated like 30 separate NumPy arrays. However, the gridpoint-wise operations are still performed in serial in this case because only a single process is available to Dask for computation. The process of splitting the underlying data into 30...
parts introduces overhead and delays the analysis, compared to the single-chunk method in green.

Similarly, introducing a multi-core Dask LocalCluster adds overhead. This can be observed through the performance of the single-chunk, multi-process implementation represented in purple in figure (1). This implementation treats the data as a single continuous array, like the single-chunk, single core run in green, but makes multiple processes available to run computation. Notably, the chunk-wise parallelism used by XCast is designed to distribute different chunks of data to different processes. Since there is only one data chunk to distribute in this case, only one process can be used and no parallelism is implemented, which results in the decreased performance.

When both multiple chunks are used and a local Dask cluster is dispatched, XCast benefits hugely from parallelism. The performance gain is generally more than enough to offset the overhead introduced. In figure (1), an implementation using multiple chunks and multiple cores which reaps the full benefits of chunk-wise parallelism is represented in brown. For optimal performance on difficult jobs, XCast can be scaled to institutional computing clusters by pointing a Dask client to a distributed task scheduler, as demonstrated by the code sample in figure (2).

```python
>>> from dask.distributed import client
>>> client = Client("127.0.0.1:8080")
>>> X, Y, T = xc.NMME_IMD_ISMR()
>>> X, Y = xc.align_chunks(X, Y, 5, 5)
>>> mlr = xc.rMultipleLinearRegression()
>>> mlr.fit(X, Y, rechunk=False)
>>> mlr.predict(X, rechunk=False)
```

Figure 2: Scaling XCast to an institutional computing cluster

3.3 Flexibility

Having been co-designed by a climate scientist and a software developer, XCast was originally tailor-made for climate forecasting. It is, however, purposefully general enough to be useful in other scientific domains which utilize gridpoint-wise data science operations. It implements a set of function decorators and a suite of generalized classes, either of which can be used to extend the functionality of traditional Python data science tools to accommodate gridded data. If a given data science method is found to be missing from XCast’s prepackaged toolkit, the user can easily use XCast’s generalized functions and classes to implement a gridpoint-wise version of the desired tool with only a few lines of code. Additionally, the types of data compatible with XCast are not limited to only a single given scientific domain. By restricting the format of the data, rather than the content, XCast allows gridpoint-wise data science operations to be applied to any four-dimensional gridded data. If a dataset can be expressed in two spatial dimensions, a ‘sample’ dimension, and a ‘feature’ dimension, it can be used with XCast.

4 Case Study: Seasonal Forecast of Indian Summer Monsoon Rainfall (ISMR)

Here, XCast is used to generate a gridpoint-wise, multi-model ensemble precipitation forecast over India. The Indian Summer Monsoon season takes place during the months of June-September, and total rainfall during this season displays a high level of interannual variability, which can make generating skillful forecasts difficult. For this reason, it is useful to communicate forecast information as tercile probabilities, indicating the probability of a “Below Normal” (BN), “Near Normal” (NN) or “Above Normal” (AN) season, as delimited by the 33rd- and 66th-climatological percentiles at each gridpoint. Additionally, Multi-Model Ensembles (MME) are well accepted as a way to improve on the forecasts of General Circulation Models (GCM) (Acharya et al., 2011). They are generally produced by averaging ensemble member outputs with equal weight, or with weight according to prior skill. There is significant interest in using machine learning-based regression techniques for MME construction because they show the potential to capture non-linear relationships (Acharya et al., 2014). To address the commonly non-gaussian nature of ISMR, a non-parametric machine learning method known as Probabilistic Output Extreme Learning Machine (POELM) is used (Wong et al., 2020).

Experimental Design

Extreme Learning Machine (ELM) is used to produce deterministic ISMR forecasts, and Probabilistic Output ELM (POELM) is used to generate probabilistic forecasts. Eleven North American Multi-Model Ensemble (NMME) General Circulation Model (GCM) forecasts, (Kirtman et al. 2013) initialized in May of each year from 1982 to 2018, are used as the predictors. Aggregated daily precipitation data from the India Meteorological Department is used as the predictand. (Rajeevan et al, 2006) XCast is used to implement linear interpolation of the predictors onto the spatial resolution of the predictand. MinMax preprocessing is applied to the ‘regridded’ predictors, and One-Hot Encoding is applied to the predictand. During One-Hot Encoding, tercile categories are defined based on the 33rd and 66th percentiles of the observed precipitation at each gridpoint. Leave-One-Year-Out cross validation is used to construct a cross validated hindcast dataset, which is then compared to the original predictand dataset in order to generate skill maps. Versions of the code implementing this case study are available in Jupyter Notebook format at https://github.com/kjhall01/monsoon/blob/main/IndiaJJAS.ipynb.

Results

Pearson Correlation and Index of Agreement (IOA) are shown to indicate the skill level of the deterministic forecasts, and the Generalized Receiver Operating Characteristics (GROC) and Rank Probability Skill Score (RPSS) visualize the skill and level of discrimination of the probabilistic forecasts. (Mason & Graham, 2002; Murphy et al., 1969) The skill scores were calculated by comparing a cross-validated Hindcast dataset with the observed values. These skill maps were calculated using a Hindcast dataset generated by taking the ensemble mean of thirty stochastically initialized Extreme Learning Machine Models.
Introducing XCast

Figure 3: Examples of Deterministic and Probabilistic Rainfall Forecasts for the 2018 JJAS season, generated with a model trained on 1982-2017.

Discussion It is clear from the forecasts in figure (3) and the skill metrics shown in Figure (4) that it is difficult to forecast Indian Summer Monsoon Rainfall. However, both relatively high Pearson Correlation and relatively high RPSS are observed in both the central northern region and the southeastern region. The similarity between the spatial distributions of deterministic forecast skill and probabilistic forecast skill serves as a reasonableness check for the ELM forecast method, and may indicate the possibility of some climatological explanation.

7 Remarks

XCast is designed to be an easy to use and flexible toolkit for climate forecasting. It is meant to be easy to learn and easy to extend for those familiar with Xarray and scikit-learn. It lets programmers and scientists access chunk-wise parallelism through Dask with significantly less overhead time investment. XCast has made it significantly easier to pursue this type of scientific inquiry by lowering barriers to the computational earth sciences like slow computation, unwieldy datasets, and opaque library APIs. In the future, XCast’s performance needs to be analyzed in a cluster computing setting. An aspirational version of XCast which manipulates big data fully out-of-core is also under development. There is certainly much more work and analysis to be done.

More information about XCast’s functionality and implementation, as well as examples and other case studies, are available in the project’s documentation (https://kjhall01.github.io/xcast), and the project’s Github repository (https://github.com/kjhall01/xcast).

ACKNOWLEDGMENTS

The authors humbly thank and acknowledge the NCAR UCAR Software Engineering Assembly for hosting the Improving Scientific Software conference, which has served as inspiration and accountability. Additionally, we thank the India Meteorological Department for the use of their daily precipitation records, and the North American Multi-Model Ensemble member GPCs for the use of their forecast data. Finally, we thank the Pangeo community for their work supporting XCast’s core dependencies, as well as the developers of XClim, ClimPred, MetPy, XSkillScore and the rest of the Python Climate Data Science Ecosystem.
REFERENCES


[13] Logan, Travis, Bourgault, Pascal, Smith, Trevor James, Huard, David, Biner, Sébastien, Labonté, Marie-Pier, Rondue-Genesse, Gabriel, Fyke, Jeremy, Aoun, Abel, Roy, Philippe, Ebbrecht, Carsten, Caron, David, Stephens, Ag, Whelan, Christopher, Low, Jwen-Fai, Keel, Tom, Lavoie, Juliette, Tanguy, Maliko, Barnes, Clair, … Quinn, Jamie. (2022). Ouranosinc/xclim: v0.35.0 (v0.35.0). Zenodo. https://doi.org/10.5281/zenodo.6407112.


Continuous Integration of METplus Use Cases
Using GitHub Actions and Docker

George McCabe
Research Applications Laboratory (RAL)
NCAR
Boulder, CO USA
mccabe@ucar.edu
(ORCID: 0000-0001-6266-9090)

Abstract - METplus is a software suite used around the world to perform consistent, reproducible Numerical Weather Prediction model verification. It consists of multiple highly configurable software components. Community contributions are encouraged to extend the capabilities. However, this flexibility makes effective testing challenging. Individual software components may be sufficiently tested but changes may inadvertently affect other components. Software dependencies required for one user’s contributions may not be available in other users’ computing environments.

The METplus team has implemented a complex continuous integration framework designed to address these challenges. METplus leverages GitHub Actions to trigger workflows that perform a variety of tasks. Over 100 use case examples are provided in the METplus GitHub repository. They are automatically tested to confirm that changes to the METplus components do not break or change existing functionality. A set of rules determine what should be run during an automated workflow. A subset of tests can be run when a developer pushes changes to a repository. The full suite of tests and logic to compare the output to truth data are run when a request is made to merge changes. A developer can also manually enable or disable testing components. This flexibility improves the development process while avoiding unnecessary execution of jobs.

Docker integrates nicely into GitHub Actions and is used to perform many useful functions of the METplus testing workflow. The MET C++ executables are installed inside a Docker container so that the desired version of the software can be easily obtained instead of installed redundantly. Each use case corresponds to an input dataset that is made available through Docker data volumes. Output is stored on DockerHub and is used in comparison tests to alert developers of unexpected differences. Docker is used to isolate testing environments for cases that require additional software dependencies.

This manuscript will (a) outline the automated testing performed through GitHub Actions for METplus, (b) describe the critical role of Docker containers in this workflow, and (c) illustrate how this approach streamlines the development process, saves time and money, and produces results that are more robust.

Keywords - Continuous integration, software, testing, containers, automation, model verification

I. BACKGROUND AND GOAL

The Model Evaluation Tools (MET) were developed by the Developmental Testbed Center (DTC) and first released in 2008. The tools were designed to perform consistent, reproducible Numerical Weather Prediction (NWP) model verification. (1) The tools have since expanded into a software suite known as METplus, which is made up of many components. MET consists of C/C++ applications used to compare forecast models with observations to compute statistics and diagnostics. METviewer is a database and display system written in Java to perform deep analysis of MET output. METexpress is another display system intended for quick analysis queries. METdbLoad loads MET output into a database that is used by both METviewer and METexpress. METcalcpy is a Python package designed to perform additional analysis of MET output such as statistical aggregation and event equalization. METplotpy is a Python package used to generate images from MET output. METviewer and, to a lesser extent, METexpress leverage METcalcpy and METplotpy for the aggregation of results and plotting. Frequent interaction occurs between these components. The METplus Wrappers are a suite of Python scripts that build commands to run one or more of the METplus components. The wrappers allow users to easily run the tools to process data for many run times without the tedious process of typing out each command or writing their own scripts to handle the command generation. Numerous calls to MET tools that previously required multiple configuration files can be executed using a single METplus configuration file.

METplus is used by forecasters, operational centers, universities, and national laboratories all over the world. Each component is highly configurable to meet the diverse needs of its users. Community contributions are also encouraged to extend the capabilities. However, this flexibility makes effective testing challenging. METplus is spread across multiple GitHub repositories so users can easily obtain only what they need. Individual software components may be sufficiently tested but changes may inadvertently break other components. Software dependencies required for one user’s contributions may not be available in other users’ computing environments. The METplus team has implemented a continuous integration framework that
utilizes GitHub Actions and Docker designed to address these challenges.

Over 100 use case examples can be found in the METplus Wrappers GitHub repository, which is available at https://github.com/dtc/higher/METplus. They are provided to demonstrate various applications of the software. Users can copy and modify the use case configuration files to perform similar verification of their data. The use cases are also used to test the software. Docker is used to set up and isolate the testing environments to run each use case.

II. DOCKER

Docker is a platform used to quickly build, test, and deploy software using containers (2). Docker images are often described as a template or blueprint for a container. Docker images are built from commands found in a text file known as a Dockerfile. These commands are read and executed by a Docker build command. This creates a computing environment by obtaining files and installing software. A Docker run command reads a Docker image to create a container. Additional commands can be run inside of it. Directories on the workstation running Docker can be mounted so that local files can be made available to the container and files inside the container can be made available locally. Docker images can also be pushed up to DockerHub, a web service provided by Docker, which makes them available to download elsewhere or use as the base of another image. Docker and DockerHub are utilized in a variety of ways to set up and run each METplus use case.

A. Obtaining Software

The MET C/C++ applications are compiled into Docker images so that official releases can be made available for users and development versions can be used for testing. The tools depend on a number of external libraries such as NetCDF. To avoid frequently rebuilding these libraries for each compilation, we maintain a Docker image that contains the required libraries on DockerHub. The FROM keyword in a Dockerfile tells Docker which image to pull to create the initial image. The MET Dockerfile pulls the base image from DockerHub then runs commands to install the MET libraries and executables. This makes it easy to install development versions of MET and run unit tests to ensure that new changes do not break any logic.

![Diagram showing how the MET C++ library dependencies are obtained to build the MET tools using Docker.](image)

**Figure 1: Diagram showing how the MET C++ library dependencies are obtained to build the MET tools using Docker.**

Nearly every METplus use case runs at least one MET executable, so the METplus Dockerfile uses the MET image as the base image. It supports optional build arguments to control which version of the MET image to use and how to obtain the METplus wrappers source code. METplus images used for testing typically use the “develop” version of MET, which contains all of the development changes that will be included in the upcoming release, to ensure that the use cases run successfully using the latest additions to MET. The METplus source code can be obtained by cloning the repository and checking out a specific branch within the image or by copying local files into the image.

B. Input Data

Each use case requires input data to run. The full set of data required to run all of the use cases is very large, so the data are divided into groups. The use cases in the METplus repository are organized by category. Each category has a corresponding dataset that contains all of the files needed to run all of the use cases in that category. These files are added to a Docker data volume that can be mounted to a Docker container that will run the use case. Each data volume is limited to 10 GB, which is the default maximum size of a Docker image. The input data volumes are stored on DockerHub and gives it a nickname that can be used to mount it to a container by adding the --volumes-from argument to a Docker run command.

C. Test Environments

Many of the METplus use cases can be run using only the METplus Docker container and the appropriate input data. However, some use cases have additional dependencies beyond the minimum requirements of the METplus wrappers. Many of the MET applications support reading non-native data formats via a call to a user-provided Python script. Instead of passing files directly to the tools, a Python script prepares and serves data to the application. These Python scripts often require additional Python packages. Other METplus components, such as METplotpy and METcalcpy, have additional Python package dependencies. Some use cases even utilize external software tools such as the GFDL Tracker used to track hurricanes. Docker images have been created to isolate the different testing environments so that only what is required by a use case is available for the test. The Python packages are installed using Conda.

The Docker images used to run the use cases are created from another Dockerfile that runs a multi-stage build. In a multi-stage build, more than one FROM statement is used. Each FROM statement pulls images from DockerHub. The last FROM statement determines the base image to use. Select files from other images can be copied into the final image. In this case, the first FROM statement pulls the image that contains the additional testing requirements and the necessary files, such as a Conda environment, are copied into the final image. This approach improves the efficiency of the tests as it is very time consuming to install all of the dependencies inside the METplus container for each test.

After the use cases have run, the output can be compared with files generated from a previous run that are known to be correct, which we refer to as the “truth” dataset. The truth data is also stored on DockerHub. The new output and the truth data directories are mounted to another Docker container that contains tools used to run
the difference test logic. The files in each directory are compared to ensure that the results have not changed. Sometimes differences are found that are expected. For example, developers may discover a bug that produces incorrect output. Fixing that bug may change the results. A newly added use case will also flag differences because the new output does not yet exist in the truth dataset. When expected differences occur, the truth dataset is regenerated, incorporating the new data for use in future tests.

III. GITHUB ACTIONS

Docker provides useful functionality to handle the complex process of testing the METplus use cases. However, there are a lot of moving parts involved, and it may be difficult to know when and how each step should be run. Fortunately, Docker integrates nicely with GitHub Actions, a continuous integration and continuous delivery (CI/CD) platform provided by GitHub (3). GitHub Actions Workflows are defined in YAML files that are checked into a GitHub repository and are automatically executed by GitHub Actions when repository events occur and the required criteria is met. Workflows consist of one or more “jobs,” which are a set of commands run serially in their own environment. Each job consists of one or more “steps” that run shell commands or execute “actions” that perform common tasks. The GitHub Actions Marketplace provides many actions written by GitHub or third party contributors that can be incorporated into a workflow (4). A commonly used action is called checkout, which is used to obtain code from a GitHub repository (5).

All workflows can be reviewed from the Actions tab of the GitHub repository website. Red, green, and yellow icons show the status of each workflow run. Clicking on a workflow run will display a summary of the run, where all of the jobs are displayed. Clicking on a job will display the status of each step within the job. Clicking on a step will display the log output for that step. The information in the logs is useful to determine the cause of a failure. GitHub Actions workflows can vary in complexity from performing small tasks like generating an automatic response to a new GitHub issue or building documentation to larger tasks like testing the METplus use cases.

Figure 2: Diagram of a METplus GitHub Actions testing workflow run that shows job dependencies. All jobs inside a box must finish running before jobs inside a connected box to the right can start running. The diagram for each run can be viewed through the GitHub web interface at https://github.com/dtccenter/METplus/actions. The YAML file that defines this workflow can be found at https://github.com/dtccenter/METplus/blob/main_v4.1/github/workflows/testing.yml.

A. Job Control

The METplus testing workflow contains rules that determine which events should trigger it. The workflow is triggered when changes are pushed to the METplus GitHub repository or a pull request is created to merge changes into a branch, unless the changes are isolated to the directory that only contains documentation. The jobs from the workflow will appear as “checks” in the GitHub pull request web interface so reviewers can easily see if the tests succeed or fail as part of the review process. Workflows can also be triggered manually or by events that occur in other repositories. For example, changes pushed to the develop branch of the MET, METcalcpy, or METplotpy repositories trigger a workflow in the respective repository that sends a signal to the METplus repository to run the testing workflow. This makes it easier to determine if a use case has broken due to changes made in another repository.

The first job of the testing workflow determines which other jobs should be run. The type of event and name of the branch determine the default behavior to run. For example, push events on a branch name that starts with “feature” will run the Python unit tests for the METplus wrappers and any use cases that have been explicitly marked to run. Push events on the “develop” branch will run all of the use cases. Pull request events will run all of the use cases and compare the output to the truth data. Push events on the “develop-ref” (development reference) branch will run all of the use cases and automatically update the truth data volumes with the output. Workflows triggered by another component such as MET will run the difference tests to catch changes caused by other components as early as possible. Keywords found in the last commit message before a push may override the default behavior. For example, a developer can add “ci-run-diff” to the last commit message before a push to their feature branch to force the difference tests to run. This allows them to test that their changes did not break anything before creating a pull request. A list of commit message keywords that can be used for this workflow can be found in the METplus Contributor’s Guide at https://metplus.readthedocs.io/en/latest/Contributors_Guide/continuous_integration.html#commit-message-keywords. The job control rules are set in variables that are output by this job and read by other jobs to determine if they should run or not.

B. Preparing Test Environments

The next job builds the METplus Docker image and pushes it up to DockerHub. Each job in a workflow is run in a fresh, isolated environment so files that were created by one job will not persist in a subsequent job. To avoid rebuilding the METplus Docker image inside each use case job, the image is pushed to DockerHub and pulled into other jobs. DockerHub login credentials are stored as secret, encrypted environment variables in the GitHub repository. The values for secret environment variables are hidden in the log files to prevent them from being exposed.

Another job uses a custom METplus GitHub Action to update the input data as needed. This action is found in its own repository, available at https://github.com/dtccenter/metplus-action-data-update. It compares files on a web server to DockerHub images on DockerHub. If the data does not exist on DockerHub or the files on the web server have been modified since the Docker data volume was last created, then new data volumes are created and pushed to
DockerHub. This functionality makes it easier for developers and scientists to add new data to use in the tests. Obtaining input data for tests from DockerHub is also more reliable than downloading the data directly from the web server. These two jobs are run simultaneously and both must finish running before the subsequent jobs are started.

C. Running Tests

Next all of the use case tests are run. All of the use cases are listed in a file that describes what is needed to run them. The cases are organized by category which determines which input dataset to use. The information for each use case includes an index, a name identifier, the path to the configuration file(s), and optionally a list of keywords that specify dependency requirements such as which Docker environment is needed to run. Python scripts parse this file and build commands to set up and run the use cases. A JSON file defines groups of use cases. Each group will run inside its own job in the workflow. The MET executables are run serially, but the use case jobs are run concurrently to speed up execution time. Each group is defined by a use case category and a list of indices, as well as a flag to tell the workflow whether or not to run the group if only select groups are to be run. If the job control rules determine that the output should be compared to truth data, then that logic is executed within the same job.

D. Analyzing Results

After all of the use case test jobs have completed, another job may run to update the truth data. The output from every use case group is obtained by this job using a GitHub Action feature known as a workflow artifact. A workflow artifact is a file or set of files produced during a workflow run. Artifacts allow data to be downloaded by users from the GitHub Actions web interface after a workflow run has completed. GitHub provides actions that can be used to upload artifacts (6). Another GitHub Action can be used to download artifacts that were created in previous jobs so these data can be accessed by other jobs (7). This workflow creates an artifact for each use case group that contains all of its output. If any use case produces an error, an artifact will be created that contains only the log files for the use cases that failed. If any differences between the output and the truth data are found, then all of the files that were flagged are copied into another artifact so they can be downloaded and reviewed.

IV. CONCLUSION

The use of Docker and GitHub Actions have provided many benefits to the METPlus project. Continuous testing our software has improved our development process. We have caught many bugs that may have otherwise slipped through the cracks and are more confident that code changes will not have unexpected side effects. This has made the team more efficient, as developers can focus on new development instead of setting up and running tests locally. It has helped define a clear process for adding new content, so external collaborators can more easily contribute to the project by adding their own use cases or even improving the software itself. The testing framework has saved the project time and money, but there is also room for improvement. The capabilities of the initial implementation were very basic compared to what is used today. The logic has been improved and expanded over time and continues to grow.

REFERENCES

A Tiered Approach to Scientific Software Quality Practices

Miranda Mundt  
Sandia National Laboratories  
Department of Software Engineering and Research  
Albuquerque, New Mexico, United States  
mmundt@sandia.gov

Wade Burgess  
Sandia National Laboratories  
Department of Software Engineering and Research  
Albuquerque, New Mexico, United States  
pwburge@sandia.gov

Dena Vigil  
Sandia National Laboratories  
Department of Software Engineering and Research  
Albuquerque, New Mexico, United States  
dmvigi@sandia.gov

ABSTRACT

In industry, software quality assurance rests upon decades of experience through which recommendations and best practices have been derived and applied. Scientific software, however, may not always be able to directly benefit from or implement these guidelines. While commercial software often has a clearly defined intended use from the beginning, including more clear requirements and generally more explored solution spaces, scientific software frequently accidentally evolves into production-level packages. That is, the majority of scientific software starts as a means to answer a fundamental research question and may gain users and stakeholders as it matures. While all scientific software can benefit from software quality best practices, those can seem unachievable to small, short-term, or newly-formed research projects - and then can be difficult to implement and maintain once the software begins to gain a following.

We introduce the concept of a flexible, tiered approach to software quality practices that allows projects of varying sizes and maturities to adhere to appropriate, tailored practices. These practices can be implemented at natural times in a project’s lifecycle such that they can be integrated into a team’s workflow and expand with the project as it moves from exploratory to production. We will cover an example tiered framework we developed for the Center for Computing Research at Sandia National Laboratories, factors that influenced its design, and potential paths for future research into tiered software quality frameworks.

CCS CONCEPTS

• Software and its engineering → Software development techniques; Software design tradeoffs; Software design engineering.

KEYWORDS

scientific software, software design, software quality, best practices

ACM Reference Format:
1 INTRODUCTION

Expansive interconnected software ecosystems have become a norm in today’s technologically-driven society, a standard from which scientific software is not exempt. As a result, software quality (e.g., reproducibility, scaleability, interoperability) has been given a stronger emphasis in scientific software development than ever before; however, industry guidelines for quality assurance are less often observed to fit the mold of scientific software development.

Scientific software development has a unique set of challenges, e.g., challenges with testing scientific software [7], short-lived or inadequate funding [14], a necessity for performance over other qualities [3]. Furthermore, when it comes to scientific software, the “software is valued insofar as it progresses the science.” [13] As noted by Villasana and Castelló, “[software quality assurance] tools are not able to adapt quickly to a small team’s needs. ... Software Quality Models ([such as ISO/IEC 9003, ISO/IEC 15504 SPICE, CMMI]) were created for big companies.” [17] To complicate this further, many domain scientists are self-taught software developers who are not directly connected into the wider web of software engineering and its practices. That is, they are domain experts (physics, mathematics, biology, etc.) who have spent their careers cultivating the professional and technical skills necessary for their specific domain. At the same time, they are often expected to develop quality software with little to no formalized software engineer training, frequently learning on the job rather than from software experts. Though some funding agencies for research and development have recently introduced requirements for software quality assurance [5], the question still remains on how to make a right-sized framework for scientific software development that is appropriately adapted and easy to adopt.

To this end, we set out to develop a software quality standardization framework for scientific software developers within the Center for Computing Research (CCR) at Sandia National Laboratories. The goal of this work was to provide a set of baseline requirements for all leads to follow on every project that scale flexibly and appropriately with the maturity, size, intended use, and risk of a software project. In the next sections, we will first provide background and motivation for the cultivation of this framework. Next we will discuss our methodology for determining the pieces of the framework, as well as the resulting information we gathered through that methodology. Following the methodology, we will display the framework as it was developed for CCR and then the potential threats to validity of the framework. Finally we present possible future avenues for continued development and formalized research into the area of scientific software quality frameworks.

2 BACKGROUND AND MOTIVATION

Software engineering as a practice is subject to the primary goal of engineering: “delivering maximum stakeholder value, while balancing the constraints of cost and schedule.” [2] That is, quality of the software as an engineered product is the main aim of any software engineer. By software quality, we follow the standard definition from the “Guide to the Software Engineering Body of Knowledge”: “[S]oftware quality is achieved by conformance to all requirements regardless of what characteristic is specified or how requirements are grouped or named.” [2]

For scientific software development, the methods to achieve software quality are not as well-explored as commercial or industry software development. There exist well-known software quality frameworks (such as ISO/IEC 25010 [6]) upon which quality guidelines can be derived; these guidelines, however, were designed with consumer software in mind. That is, at the onset of development, the intended use is clear - there will be a consumer who will act as the end-user around whom the software is designed. In the realm of scientific software development, however, the end-user is frequently the research scientist themselves, and the software creation is more ad-hoc as a means to answer a question as opposed to be distributed to a wider audience. Additionally, quality of cutting-edge systems (e.g., quantum computers) remains an open question, creating further barriers to applying general industry standards.

In evaluating software quality practices across projects within the Center for Computing Research (CCR) at Sandia National Laboratories, it became apparent that for projects without stakeholder-imposed quality guidelines, each project lead was left to determine their own definition and guidelines for quality. This practice, while not intrinsically problematic, did present an issue of a lack of standardization which, considering the varied backgrounds of domain scientists and the breadth of scientific software projects, led to quality activities frequently not being based on standard better practice guidelines; rather, there were more ad-hoc than intentionally designed. The quality focus for many of these domain experts was on the research results, and in terms of software, many were unaware of standard guiding principles for good software quality practices. An important consideration in this area is the eventual intended use of the software. We must consider the case where the developer is the primary user (e.g., the software was created only for that specific domain scientist) as different from the case where there may exist stakeholders and other potential end-users in the future.

In an attempt to address this concern, we as members of the Department of Software Engineering and Research endeavored to create a software quality standardization document for use by software developers within CCR. This set of guidelines was developed through a series of interviews, surveys, and literature reviews aimed specifically on aligning suggested practices with (1) the values of the organization’s leadership; (2) the unique environment of scientific software development as it differs from industry; and (3) ease of integration into existing workflows.

3 METHODOLOGY

The creation of the resulting framework relied on three main sources of information:

(1) Interviews. We interviewed members of leadership within CCR at Sandia National Laboratories with the goal of gathering data about values and views on software quality. There were a total of 12 leadership members interviewed, each a manager of an individual department. Interviews were conducted in-person and captured via detailed notes.

(2) Surveys. We created and distributed a survey to project leads to request information about current quality practices within
their software projects. This survey was distributed through a web-based form, which was completed anonymously.1

(3) Literature review. We engaged a colleague from the Department of Software Engineering and Research to conduct a rapid literature review based on the motivating questions, “Do different teams work better under different sets of quality standards? How do we right-size a software quality model?”

The methodology for each source is discussed below, whereas results can be found in Section 4.

3.1 Interviews
We conducted a short-term project to investigate software quality incentivization. The goal of this project was to determine factors that led to incentivizing software quality practices, that is, turning software quality from a “check-the-box” activity into a culturally integrated practice. A recurring theme during this study was the necessity for leadership buy-in and advocacy [12], that is, having an influential and trusted advocate for software quality practices makes others more likely to adopt them.

We used this principle to choose our interview candidates. In order to ensure strong advocates with appropriate ranking within the organization, it was essential to gather and incorporate data on the value system of those in leadership as it related to software quality. We opted to approach the entire population of twelve department managers (also known as "Level One (L1) managers") as members of the overarching leadership team who would have an intrinsic knowledge of their respective departments’ structure and projects. To achieve this, we set up one-hour in-person meetings with the current manager from each of the twelve individual departments within CCR. As part of every interview, we provided each participant with the motivation and ultimate goal of this effort, as well as a rough timeline.

The format of the interview was intentionally designed to be open-ended to allow for candid discussion as opposed to a pure “question-and-answer” format. Ultimately, however, we aimed to gather the following information from each participant:

• Q1. What is your definition of software quality?
• Q2. Based on the type of software that your department generates, what do you consider to be your process for quality?
• Q3. Assuming the existence of such a thing, what would be your trigger for a software project to transition to a new set of quality practices?
• Q4. How do your customers or stakeholders enforce quality?
• Q5. How do you measure the success of a project within your department?

Answers to these questions, as well as other topics that came up during the open discussion, were captured in digital notes with access restricted to only the authors. The notes were then summarized and sent via email to the participants to ensure clarity of the participant’s views. For any discrepancies, the notes were corrected and sent back to the participant, to be iterated upon until agreed upon by both parties.

The interviews and surveys are not considered human subjects research because the questions are designed to serve a business need, i.e., requirements gathering aimed at the creation of a standardized software quality framework.

3.2 Surveys
The body of literature surrounding individual team quality practices within scientific software development is extremely sparse. That being said, motivation in software engineering is a more comprehensively explored topic, which is impressive given its difficulty to quantify. As noted by Verner et al. [16] in a recent study, for example, software teams are more likely to engage in quality practices given the right tools, techniques, and awareness; that is, if the data about good quality practices is made both readily available and easy to use, teams are more likely to adopt them and, as a result, raise the quality of their software.

Juxtaposed, however, is the fact that teams often are unaware of the current state of their software [4]. Keeping this fact in mind, we chose to survey project leads for software projects within CCR to gather information about the current state of their software quality practices and to reduce the likelihood of unfounded assumptions and biases leaking into the final framework. The following email was sent to all members of CCR:

The [Department of Software Engineering & Research] is creating a standardized approach to software quality for those projects that don’t have quality standards predefined by a stakeholder. We have been developing a document to reflect this recommendation, and we would like to hear from you. This document is meant to be right-sized for [CCR], and we cannot do that without your help. This is your chance to be heard! If you are a [project lead] for a project, please fill out our quick [CCR] Project Software Quality Survey. It takes only a few minutes and will help us understand the collective values for software in [CCR], as well as the current state of software quality.

The survey was completely optional, and all data was collected anonymously (with the option to supply an email, should the participant decide to do so) via a web-based form. The full set of questions can be found in Appendix A.

The first page of the survey focused primarily on metadata about the project itself: (1) project maturity, (2) team size, (3) time allocation to the project, and (4) percentage of team members with some form of formalized training in software engineering. This data allowed us to directly compare the current practices of a software project with its available personnel assets.

The second page of the survey queried the participant on current practices and perceived importance of value based on the software’s intended use. Figure 2 displays one such query. This table asked the participant to rank the importance of a series of terms or activities to their project. The activities range from pure software engineering activities (e.g., software design, regular verification testing) to more high-level considerations (e.g., stakeholder satisfaction) to research activities (e.g., publications). Note that the question specifically asks, “Given your project’s intended use...” As explained in Section 2, an important consideration for us was the intended use of the project. Based on our experience as a department of Research Software Engineers [1] and the case studies conducted by Carver et. al [3], we made the assumption that for software entirely created for a researcher’s own purpose (i.e., the creator is the sole user), there would be not only a lower necessity for more rigorous quality...
assurance practices (such as end-user documentation), but also a stronger resistance to the adoption of such practices.

Given your project’s intended use, please rate the importance of the following:

<table>
<thead>
<tr>
<th></th>
<th>1 - N/A</th>
<th>2 - Not Important</th>
<th>3 - Somewhat Important</th>
<th>4 - Important</th>
<th>5 - Very Important</th>
</tr>
</thead>
<tbody>
<tr>
<td>Publications</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Presentation material</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Research answers</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Reproducibility</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Replicability</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Version control</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Stakeholder specifications/requirements</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Stakeholder satisfaction</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Software architecture</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Software design</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Software development</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Software release/deployment</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Software stability</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Software extensibility/future growth potential</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Regular verification testing</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
</tbody>
</table>

Figure 2: Quality practices survey for Project Leads

Project leads were given two weeks in which to complete the survey. The survey had an estimated 7% response rate (14 out of roughly 200 staff members). Afterwards, the data was exported for further analysis (results to be discussed in Section 4).

3.3 Literature Review

We engaged our colleague Reed Milewicz from the Department of Software Engineering and Research to conduct a rapid literature review to assist in the framework creation. According to his whitepaper publication, “[a] rapid review protocol is a systemic, time-boxed literature review designed to deliver evidence in a timely and accessible way.” [10] Generally, these reviews are motivated by practical problems; that is, they are designed to provide immediately applicable suggestions, condensed in a short time frame such that action can be taken quickly.

In the case of this effort, the two unique research questions we wished to explore were:

- **RQ1.** Do different teams work better under different sets of quality standards?
- **RQ2.** How do we **right-size** a software quality model?

To complete this rapid review, Milewicz used the search terms described in Table 1. Full details on the findings of the rapid review will be discussed in the next section.

4 RESULTS

The results of the above methodologies for information gathering are summarized below.

4.1 Interviews

As noted above, all L1 managers for each of the twelve individual departments within CCR were interviewed. The interviews were open-ended to allow candid conversation so as to gauge differences and values of each manager with respect to their department. Ultimately, they resulted in three main themes.

**Fundamental research.** The first overarching theme from the interviews with management was the emphasis on fundamental research. One participant noted:

> It’s clear that we are not a software development shop - we are a research development shop.

To that end, software quality was indeed seen as valuable insofar as it could further research and promote the trustworthiness of the research results. The main goal of the majority of projects within CCR is the research results (answers to questions, publications, presentation material), not the software which enables that goal. This value trickles into hiring considerations as well, where more often than not an applicant is selected based on their research ability as opposed to their software ability, thus perpetuating the culture of self-taught software development.

**Triggers for more rigor.** Nearly all participants in their interviews discussed potential “triggers” in a software project’s life that would require the introduction of more formalized rigor. For example, the introduction of users or stakeholders would cause an immediate necessity for rigorous quality practices as there would be an implied sense of required maintenance and support. One participant spoke from their own experience:

> Until we really had a sponsor that demanded a higher level of rigor (e.g., process documentation, traceability), there had to be a clear benefit for all of those [in order for us to do them].

The interviewees perceived that, given the limited time each staff member actually has to dedicate to software development activities, that time needs to be carefully spent. For a project that is new and trying to prove a novel concept, the rigor behind software quality is far overshadowed by the rigor required for the validation of research results - a process which can be achieved without perfect quality software. As the project matures, however, that scale can tip in the opposite direction. In other words:

> I’m a big believer in having just the amount of process you need to get what you need to get done.

Commonly paired with this was to comment that more rigor requires more expertise. To that end, the biggest help to any existing project would be “access to a complimentary, dedicated software engineer,” particularly when the software is going through a maturity transition. Before such a transition, however, the software often exists to answer a research question and only requires a basic amount of software quality rigor and a larger dedication to validation to ensure the resulting answers are trustworthy and align with the current state of science.

**Non-prescriptive guidelines.** The final common thread from most participants was the desire for less prescriptive guidelines. Many projects within CCR fall under the Advanced Simulation and Computing (ASC) program and its associated Software Quality Plan [15]. As stated by one of the interview participants:
When I had my first ASC software quality assessment, one of the reviewers told me, “We’re not here to expose external processes for you. You clearly get work done - therefore, you have processes.”

The intent behind this plan is to allow projects to define their own processes while providing guidelines for what those processes should be achieving. Rather than explicitly stating the exact tools and procedures to use, the ASC Quality Plan outlines general process areas which must be considered and addressed.

Within CCR, cutting-edge computing research is being conducted every day. One manager said of his department: “Some of the projects I have now are in a new computing region, so in some cases, I don’t even know what software quality means for some of those devices. It’s a research topic of itself.”

If software quality is still an open question, we surmised, then requiring extremely prescriptive practices was unreasonable. Additionally, because most members of CCR are at least familiar with the ASC Software Quality Plan, we hypothesized that alignment to that plan would be familiar and likely to be adopted.

4.2 Surveys

The results from the first page of the survey (the metadata) are depicted in Table 2.

For the project leads who responded, we observed trends that match those suggested in literature surrounding scientific software development. The team sizes are generally quite small (three or fewer) and the teams have a small percentage of members who have received any sort of formalized training in software engineering.

As for the second page of questions (Figure 2), the overall consensus regarding non-software end products was the essential importance of research answers and presentation material. Surprisingly for a research center, presentation material outranked publications and research materials in level of importance. We consider that this could be a result of the lack of response from proof-of-concept project leads or of the fundamental mission of Sandia National Laboratories - that is, work completed needs to meet mission milestones first and foremost, whereas publications can always be created later. (See Table 3.)

The full table of responses for the software-related activities is not represented here. Instead, we identified three main categories of activities - software development life cycle (SDLC), testing, and user and stakeholder considerations. The items that fell within each category are as follows:

(1) SDLC. Software architecture; software design; software development; software release/deployment; software stability; software extensibility.

(2) Testing. Regular verification testing; regular validation testing; regular functionality testing; regular unit testing.

(3) User and Stakeholder (U&S) Considerations. Stakeholder specifications and requirements; stakeholder satisfaction; user experience (installation); user experience (usage); maintenance and support.

As seen in Table 4, SDLC activities were rated on the higher end of importance. Particularly, though not reflected in the summarized data, there was a large emphasis on the software architecture and design activities. The other two categories have a fairly even distribution between “Somewhat important” and “Very important.” Of note in the Testing category, there was a higher importance given to functionality and unit testing over verification and validation testing.
Table 4: Perceived importance of software development activities.

<table>
<thead>
<tr>
<th>Perceived Importance</th>
<th>SDLC</th>
<th>Testing</th>
<th>U&amp;S Considerations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not important</td>
<td>4.8%</td>
<td>9.3%</td>
<td>9.2%</td>
</tr>
<tr>
<td>Somewhat important</td>
<td>24.1%</td>
<td>37%</td>
<td>29.2%</td>
</tr>
<tr>
<td>Important</td>
<td>44.6%</td>
<td>27.8%</td>
<td>35.4%</td>
</tr>
<tr>
<td>Very important</td>
<td>26.5%</td>
<td>25.9%</td>
<td>26.2%</td>
</tr>
</tbody>
</table>

These results supported many of the views expressed by the manager interviews. That is, presentation material and research answers were of clear importance to scientific software project leads as the end-product of their work. As for the enterprise of developing scientific software, SDLC activities were viewed as important; however, the small number of team members with formalized software engineering training was a cause of concern for us.

### 4.3 Literature Review

The outcomes identified by Milewicz were a synthesis of results of seven scientific primary and secondary studies [9]. The outcome of this synthesis was reflected in six main categories of relevant factors which represent different facets of software quality motivation, each of which (customer excluded) had at least one accompanying key takeaway. The details are summarized below:

1. **Individual.** Developers as individuals bring unique skills and perspectives to a project. This includes not only a familiarity with the problem domain or training in software development, but also motivations and past experiences.
   - **Takeaway.** A scientific software developer who has received formalized training in software development will ultimately create higher quality code in a more productive manner.

2. **Team.** A development team consists of several members working together on a software package with collaborative intent. This includes a sense of common identity, clear goals, and the rate of turnover.
   - **Takeaway.** Good teaming enables productivity and quality. That is, when team processes and workflows are clearly defined and followed, teams will be more productive, create higher quality code, and be able to support new development.

3. **Organizational.** An organization represents a shared value system and understanding of business goals and has sway over the direction of its projects. This includes support and commitment from upper management, budget allocations, and relationships to neighboring projects.
   - **Takeaway.** In order to achieve productivity and quality, a project must be in alignment with the organization’s values and processes, but the organization or funding source must also value quality and allocate funding towards quality activities.

4. **Technology.** Tools for software quality practices supply efficient ways to manage those activities. This includes the use of software development tools such as version control, issue tracking, and code analysis.
   - **Takeaway.** The right tools used in the right way can improve quality. That is, technology can both enable and inhibit quality, depending on usage.

5. **Process.** Processes exist to deterministically designate the steps taken for a particular task. This includes the use of development methodologies, whether well-defined or ad-hoc, and the extent to which the project is committed to using those methods.
   - **Takeaway.** Similar to the takeaway from the Team category, adherence to a managed and well-defined process for software development is likely to result in quality and productivity.

6. **Customer.** A customer may be a user or a stakeholder and can influence the direction of a project. This includes the frequency of changes in requirements, the extent of user involvement in the development, and users’ resistance to change.

An overarching message from this review was that software quality itself drives productivity. That is, software which is already of high quality will enable better productivity and will positively impact a developer’s ability to efficiently change the software.

This rapid review enabled us to ask appropriate questions in the interviews and survey as well as consider each factor in the creation of the desired framework, discussed in more detail in the next section.

### 5 IMPLEMENTATION

Using the information described in Sections 3 and 4, we produced a framework for CCR comprised of four different tiers for projects of different maturity. The intent is for a project lead to compare the characteristics of their project to those listed in Table 5 to determine the appropriate tier. Then, having determined the tier, the project lead adheres to the accompanying requirements and/or recommendations.

As the project matures and moves into higher tiers, new requirements and recommendations are then applied to the project. The software quality activities represented in each new tier are designed to be naturally aligned with the new level of maturity and match the level of rigor deemed necessary for that maturity. That is, version control is a Tier 1 requirement due to its basic necessity for all software projects. An established release process, however, would not be necessary until a project significantly matures – that is, becomes larger, more established, and has stakeholders. In this way, the framework aims to impose minimal “extra” work on developers when it is unnecessary.

Note from Table 5 that some recommendations become requirements at higher tiers (e.g., team policies and procedures). Integrated into this framework is the concept of “intended use.” We consider two different cases of intended use:

1. **To be used by a domain expert software developer (and perhaps a very small set of colleagues) to answer research questions.**
2. **To be shared with other users (internally or externally to the organization), and by implication, requires more formalized maintenance and support.**
For (1), a project would be expected to implement only requirements for the appropriate tier. For (2), however, a project would be encouraged to also implement the recommendations (by means of introducing quality activities earlier in the software’s life).

With regards to the requirements, they are left intentionally non-prescriptive (e.g., do not specify a particular tool or methodology). This allows the adaptation of activities to a particular team’s preferred workflow and current tools, an essential requirement noted from the interviews and surveys. The framework also only applies to those projects for which a stakeholder has not defined a software quality standard. That is, if a software project takes on a stakeholder that has well-defined requirements for quality activities, that would replace this framework. In that way, the framework acts as a baseline for all software developed within CCR but which can be removed if another framework applies, thus minimizing the complexity of requirements.

A fundamental offering in this framework is the availability of supporting software engineers. The Department of Software Engineering and Research is a team comprised of Research Software Engineers [1] who are made generally available for consultations and integration into software projects. This department (described in more detail by Milewicz et al. [11]) aims to provide formal software engineering support to all members of CCR with the goal of improving software development practices, and as the surveys show, the majority of scientific software projects do not contain members with formalized training. In this way, the Department of Software Engineering and Research aims to fill the gap of knowledge and improve computational outputs. To enable further self-education and asynchronous assistance, project leads were also provided a comprehensive list of resources and definitions for terms and activities (not included here).

Prior to its release, the framework was inspected and reviewed by members of two third-party quality assurance teams within Sandia National Laboratories: Jennifer Turgeon of the ASC Software Quality Plan and Heidi Jones, Quality Engineer from the Assurance Excellence organization. These two staff members independently reviewed the framework with the intent to identify potential gaps from the organizationally-required policies, as well as potential improvements which could benefit the members of CCR. The result of these reviews brought the integration of risk considerations into tier determination; that is, both reviewers noted that within the national laboratory environment, even if most of the work completed by CCR staff is meant to be open science, risk level would be wise to include in level of rigor. This addition can be seen in the Tier Characteristics column in Table 5.

6 THREATS TO VALIDITY

There were several gaps or threats to the validity of the methodology used to generate this framework.

Interviews. Though the entire population of L1 managers were queried, no interviews were conducted with higher level managers. We decided to focus on the leadership team closest to the scientific developers; however, there may have been different views which may have represented the organizational values surrounding software quality assurance. The resulting framework was geared towards the L1 managers’ value system, and as such, may have lacked enough overarching organizational values. Additionally, because the interviews were open-ended as opposed to semi-structured, there was plenty of room for interpretation and non-concrete answers. For example, some of the departments are structured such that the answers to several of the target questions would be, “It depends,” which makes quantitative extraction of data difficult.

Surveys. An unfortunate issue with the responses, as can be seen in the Table 2, is the lack of response from those projects in the least mature category. Though there was a respectable response rate of 7%, no project leads with proof of concept software chose to complete the survey, so there is a complete gap in those projects that require the least amount of rigor. As a result, there is a high likelihood that there is a skew in the data towards practices and rigor applied to more mature projects (e.g., automated testing) and away from fundamental research practices (e.g., publications). Additionally, we received no information on the current practices for those lowest-level projects, which introduces the risk that the Tier 1 requirements and recommendations may not be appropriately right-sized.

Literature review. A significant threat to validity was the use of a rapid review rather than a full systemic review. While rapid reviews allow fast turnaround, they do omit certain steps, such as limiting the literature search and lowering the quality appraisal requirements. This presents the possibility that certain topics were overlooked or incorrectly summarized, though we are fairly confident that the selected literature and resulting outcomes are reasonably reliable.

Implementation. This framework was released in late 2020 to the members of CCR. The standardization has been softly enforced since then; that is, the adherence to the standardization is currently an honor-based system, intended to be encouraged by the L1 managers. Additionally, this framework was released at the peak of COVID restrictions. We did not make any changes or take this into account in the release process which likely affected its adoption rate. At the end of 2021, we released a tool (internal to Sandia National Laboratories) to help projects track their implementation of the requirements and recommendations, as well as rank their implementation of the requirements such that they will be able to track progress over time, but as of this publication, use of this tool is also not enforced. As a result, we do not have reliable data on the adoption rate and therefore cannot speak to the efficacy of the framework overall. Additionally, the development and implementation is highly dependent on the owning organization’s value system. The methodology and ultimate framework is only possible insofar as leadership values and backs the system. As such, it may be difficult to directly recreate this process.

7 FUTURE WORK

During the development phase, feedback on the framework was received by the members of CCR (the framework is also updated yearly, based on those recommendations and changes in organizational values). Future studies into the efficacy of such an approach to framework creation would be beneficial, as well as case studies into its application in other scenarios outside of national laboratories (e.g., academia). Because a key factor in software quality
motivation is alignment with the owning organization, the presented framework is likely not a perfect fit for all other national laboratories or academic departments. Ideally, the methodology through which the framework was derived can be reproduced and adapted in other environments and yield similar guidelines for organization members.

As noted in Section 5, we engaged several members of different quality assurance teams to identify organizational gaps. Such work could be extended to map the above framework to other commonly recognized and accepted measures of maturity or quality. For example, we determined the tier characteristics from the characteristics valued by the owning organization. Mapping these tiers to technology readiness levels (TRLs), more commonly understood among research scientists, could be a useful alteration. As another example, future work could be dedicated to comparing the requirements and recommendations to a common standard, such as ISO/IEC 25010 [6].

We also intend to augment this framework to include research paper quality guidelines, an effort already introduced to CCR by a Senior Scientist. Exploration into the integration of research paper and software quality guidelines into a single framework is a promising avenue of research.

8 CONCLUSION

Scientific software development benefits from the decades of industrial experience through which recommendations and best practices in software quality have been derived, but only insofar as that experience is properly adapted for scientific software teams. Through the use of interviews, surveys, and a literature review, we explored the question of how to right-size a software quality model and derived a framework for use in the Center for Computing Research at Sandia National Laboratories.

Incorporated into this framework is a tiered approach to software quality activities in which lower tier projects (e.g., newer, smaller, short-lived) require less rigor and have more basic requirements (such as version control and requirements development). As the project matures and enters higher tiers, activities are introduced which align with the new stage of the project, thus making software quality a harmonious portion of the software development life cycle for scientific software engineers.

While this framework is specifically designed for the organizational value system which exists at Sandia National Laboratories, the methodology can be reproduced for other organizations, yielding customized frameworks.
ACKNOWLEDGMENTS

We in particular want to thank our colleagues within Sandia National Laboratories for their contributions to this work: Reed Milewicz for his rapid review, Jennifer Turgeson and Heidi Jones for their independent reviews, and Scott Collis for his constant support, advocacy, and commitment to software quality throughout the Center for Computing Research.

Special thanks also to Katerina Limpitsouni [8] for use of undraw.co image in Figure 1.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy’s National Nuclear Security Administration under contract DE-NA0003525, SAND2022-10118 C.

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

REFERENCES


A SURVEY QUESTIONS

A.1 Metadata Questions

(1) Open-ended. List the name of your project.

(2) Yes/No. Does your project involve the creation of any code or software (scripts, algorithms, tests, single-use codes, team-developed code bases, open-source software)?

(3) Multiple choice. Software projects in CCR can range from experimental prototypes to mature, production-oriented codebases. Which of the following best describes your project?

• Early research stages
• Very exploratory (e.g., proof of principle scripts)
• Somewhat exploratory (e.g., a working prototype)
• Somewhat productionized (e.g., ready for production; a somewhat stable but actively evolving project)
• Very productionized (e.g., a mature, regularly maintained codebase)

(4) Multiple choice. How many team members are on your project?

1
2-3
4-5
6-10
11-15
15+

(5) Multiple choice. How many full-time employees are allocated to the project?

• Less than 0.5
• 0.5 to 1.49
• 1.5 to 2.49
• 2.5 to 3.49
• 3.5 to 4.49
• 4.5 to 5.49
• 5.5+

(6) Multiple choice. What percentage of your team has formal background or training in software development?

• 0-24%
• 25-49%
• 50-74%
• 75-100%

A.2 Value Queries

A visual of this question is shown in Figure 2. For the following options, survey participants were asked to rank the importance of each (considering the intended use of their project) on a scale of 1-5 (1 - Not Applicable, 2 - Not Important, 3 - Somewhat Important, 4 - Important, 5 - Very Important):

• Publications
• Presentation material
• Research answers
• Reproducibility
• Replicability
• Version control
• Stakeholder specifications/requirements

2If “No”, all other questions are skipped.
• Stakeholder satisfaction
• Software architecture
• Software design
• Software development
• Software release/deployment
• Software stability
• Software extensibility/future growth potential
• Regular verification testing
• Regular validation testing
• Regular functionality testing
• Regular unit testing
• User experience (installation)
• User experience (usage)
• Maintenance and support

We also chose to add a value query specifically geared towards documentation. The participants were given the prompt, "Given the current state of your project, what documentation do you think is important and what documentation currently exists?" As with the previous question, they were asked to rank the following types of documentation on a scale of 1-5 (1 - Not Applicable, 2 - Not Important and Doesn’t Exist, 3 - Not Important and Exists, 4 - Important and Doesn’t Exist, 5 - Important and Does Exist):

- Team policies and procedures
- Software architecture/design
- Coding style guidelines/standards
- Technical documentation
- Developer documentation
- Software testing process

• Software build and deployment process
• End-user how-to documentation
• Maintainer how-to documentation

A.3 Current Practices
The final question in the survey related to current practices. The participant was prompted to check all options which finished the statement, "My project currently...":

- Uses a version control system
- Conducts verification testing
- Conducts functionality testing
- Conducts unit testing
- Conducts local/manual testing
- Conducts automated testing
- Conducts automated release/deployment
- Conducts user support
- Tracks issues/bugs
- Practices requirements management
- Practices project management
- Practices release management
- Practices code design
- Practices code review
- Depends on internal collaboration (within CCR)
- Depends on internal collaboration (within Sandia National Laboratories)
- Depends on external collaboration (outside of Sandia National Laboratories)
Communication Overlap In Large-Scale Spectral Transforms

Dmitry Pekurovsky

San Diego Supercomputer Center
University of California, San Diego
La Jolla, California, USA
dpekurovsky@uscd.edu

ABSTRACT

P3DFFT++ is a highly adaptable open source framework for multidimensional Fast Fourier Transforms and related spectral algorithms. This class of algorithms covers a large number of computational domains. Spectral algorithms are quite difficult to scale on large HPC systems. Hiding communication latency, at least partially, is one way to reduce time to solution, and is one of the goals of this work. In addition, this work aims to implement the algorithms so they can run on heterogeneous platforms with GPUs, and do so in a highly portable, user-friendly manner. In this context we implement and evaluate overlapping computation with network communication, in a way suitable for future work on overlapping data transfer to/from GPU. Eventually the framework will have a unified API for running on CPUs and GPUs. Our benchmarking results demonstrate potential of communication overlap to improve performance by up to 22%.

1 Introduction

Spectral Transform is a type of algorithms commonly used in a variety of areas of computational science and engineering to simulate a wide range of phenomena. Examples include Direct Numerical Simulations of turbulence [1-7], astrophysics [8-10], plasma physics [11], material science [12,13], space physics [14] and seismic simulations [15], to name just a few. This class of algorithms includes, but is not limited to, Fast Fourier Transforms (FFTs). Due to their ubiquitous nature, they are found in many third-party packages, such as numerical libraries (briefly reviewed in Section 2 below).

Spectral transforms are challenging to implement efficiently on large-scale High Performance Computing (HPC) systems. The challenge of extracting a good performance out of FFTs at large scale, for example, is well-studied and has to do with dependence on the system’s bisection bandwidth, as well as on-node memory bandwidth [16-18]. They are already a major bottleneck for many important applications, responsible for a large number of compute cycles. Performance challenges are likely to become exacerbated as we approach the Exascale.

This paper describes continuing work on P3DFFT++ (http://www.p3dfft.net), an open source general purpose library for Spectral Transforms in 3 and eventually 4 dimensions [19], following earlier work on P3DFFT, a library for scalable Fourier Transforms [20]. It aims to provide an efficient, scalable solution for Spectral Transforms beyond FFT, in an easy-to-use, portable fashion. Our previous year proceedings paper [21] described the motivation and design of the library, including the basic software framework and the extremely flexible data structure that can support a wide range of usage scenarios.

In this paper we continue this work to focus on performance aspects, in particular the overlap of communication with computation through the use of nonblocking communication. We describe several ways that this can be achieved in an algorithm such as 3D spectral transform, and how it is implemented in P3DFFT++. We present preliminary performance results from our ongoing efforts, which demonstrate some promise of our approach. The hope is that eventually success in overlapping communication with computation will lead to significant performance advantage of our implementation, compared to other packages that do not employ an overlap. This will lead to a more efficient library package that will translate to significant decrease in throughput time for the end user, as well as saved cycles on high-end HPC platforms.

2 Previous work and problem definition

In this work we refer to spectral-like transforms when we mention any multi-dimensional transform algorithm on a structured grid that has the following properties:

1. It can be reduced to a sequence of 1D transforms for an entire array, one for each dimension, independent of other dimensions (e.g. 1D FFT).

2. Each such 1D transform is compute and memory-bandwidth intensive. In terms of data decomposition, it is best to have all data in that dimension to reside locally in memory for each core/task. This allows us to avoid exchanging data within each 1D transform, which would be extremely expensive.
In order to have such local data access, there needs to be a reshuffling of the data between the 1D stages. This is illustrated in Fig. 1, where a typical 3D FFT algorithm with 2D processor decomposition is broken down into steps. (Note that 2D decomposition is critical for larger scale use of spectral transforms. 1D decomposition does not scale beyond \( N \) cores/tasks, where \( N \) is the linear grid size of the 3D array \( N^3 \). 1D decomposition is a limitation of several earlier libraries, such as FFTW.) Therefore a 3D transform comprises of three 1D transforms, interspersed with two all-to-all exchanges in subcommunicator groups. This logic is followed in P3DFFT, an earlier incarnation of P3DFFT++, and the current version of P3DFFT++ [19]. 2D decomposition, in principle, allows the algorithm to scale up to \( N^2 \) cores/tasks.

While P3DFFT++ follows the same logic, it allows for a much wider set of data structures and transform types than in P3DFFT. It is written in C++/MPI, with Fortran and C interfaces. It utilizes established 1D FFT implementations such as FFTW [22], ESSL [23] or cuFFT [24]. The software framework is object-oriented and can be easily expanded. There is an early CUDA/GPU version, in addition to the CPU version, which is more established. The library implements real-to-complex and complex-to-complex FFTs, as well as sine/cosine transforms. It is extensible for other transform types, for example wavelets and high-order finite difference schemes. It features a very general data layout, which should be sufficient to map to the layout of almost any user program. Also, quite importantly for mixed precision applications, it works with both single and double precision data. The package includes example programs in all three languages (C/C++/Fortran), as well as full documentation. P3DFFT++ has been found to scale well up to 32k cores and beyond, provided adequate hardware support (see Fig. 2). An earlier generation of the library, P3DFFT, was found to scale up to 512k cores [5], and the same scaling is expected for P3DFFT++.

![Figure 1: 3D FFT implementation with 2D decomposition typically involves a sequence of 1D transforms in X, Y and Z dimensions, interspersed with two sub-communicator all-to-all exchanges.](image)

A number of other 3D FFT/spectral transforms packages have been published in the last decade, confirming a great interest in this class of algorithms. They include both open source, third-party libraries as well as parts of open and proprietary codes. While most implementations share the basic structure of 3D algorithm shown in Fig. 1, there are many variations in terms of types of algorithms covered, data structures assumed, and performance features incorporated. It would go beyond the scope of this paper to do a thorough review of existing packages, so we will mention only a few libraries that are most recent and noteworthy in our opinion.

heFFTe [25] is a new powerful library from UTK group. It came about as a part of DOE’s Exascale Computing Project. It is written using modern C++11 features. It provides both CPU and GPU implementation. It provides an option for 3D decomposition and implements real-to-complex and complex-to-complex transforms. This library does not seem to support other spectral transforms, and, like most libraries, is designed for a fixed data layout.

FFTE [26] is another implementation for both CPU and GPU. It has been in continuous development for a number of years. Its latest version provides support for NVIDIA GPUs only through PGI compiler. In addition, it has limited features – for example, only grids of certain sizes and MPI rank counts are supported.

AccFFT [27] is a library implementing real-to-complex 3DFFT on NVIDIA GPUs. Its last release was in 2014, and it is no longer supported.

SWFFT [28] is a stand-alone 3D FFT CPU library derived from Hardware/Hybrid Accelerated Cosmology Code (HACC) from Argonne [29]. It is one of the few libraries supporting 3D decomposition. It seems to support only complex FFTs. Also, it seems to have restrictions in sizes of transforms and decompositions.

FLUPS [30] is a library for solving Poisson unbounded problems using spectral methods. It provides sine/cosine transforms, in addition to FFTs, in combinations for three dimensions. This makes it (along with P3DFFT++) more general than other...
libraries. It has hybrid MPI/OpenMP implementation and is capable of using nonblocking communication. It lacks advanced features, such as general data layout. It also does not have a GPU implementation.

NB3DFFT [31] is one of very few libraries implementing overlap of communication with computation through nonblocking communication. It was last released in 2015 and is limited in the features it supports. This short overview, hopefully, gives the reader an idea of the space of features spanned by modern spectral transforms libraries. While no single library can be expected to have the best solution for any problem and platform, P3DFFT++ aims to maximize the extent of both portability, usability and performance. The latter aspect is an extremely important one for practical use on modern systems, as mentioned previously, and will be the subject of the remainder of this paper.

3 Nonblocking Communication

The idea of overlapping communication with computation is not new, however it is not often found in spectral transforms implementations. One reason for this may have to do with not having an obvious way to stagger the stages of the algorithm to overlap communicating with computational stages. After all, the stages of the algorithm depend on each other and are supposed to be done in sequence. We see a couple of ways to solve this problem. One situation when this becomes possible is when the original problem has several independent variables to be transformed (for example, velocity components). In such case the algorithm execution for these variables can be staggered, so that communication for one variables is happening while another variable goes through a computation phase (see Fig. 3).

Another way to get traction with the overlap idea is to divide the local grids into a number of segments, or chunks, and stagger the stages of the algorithm over these chunks so as to overlap communication of one chunk with computation of the next one. The splitting of local arrays translates into smaller message sizes, and thus can potentially reduce the efficiency of communication. This is undesirable and less effective than expressing the overlap through multiple variables, however this may be the only way to introduce overlap in case when there is only one variable to be transformed.

While work is underway to implement multi-variable transforms in P3DFFT++, we can refer the reader to our past work on P3DFFT [32], where we have shown that the benefit of overlap through nonblocking communication in some cases can be as high as 27% overall performance improvement (see Fig. 4). The remainder of this section deals with single-variable overlapped algorithm through splitting the buffers.

Figure 4. Results from previous work [32] comparing performance from blocking vs. nonblocking (overlapped) implementation of P3DFFT. Time of solution is plotted on the vertical axis (less is better). Different nonblocking data sets correspond to different MPI implementation strategies, rather than P3DFFT implementations. For more details see Ref. [32].

3.1 Single-variable overlap

As mentioned above, in order to implement overlap in a single-variable algorithm, it is necessary to split the local buffers. For simplicity, we split them into $N_c$ equal parts. The best value for $N_c$ depends on a number of factors, such as the system hardware, system software, problem size and data type. The algorithm then becomes a modification of Fig. 1, where computation of one segment is overlapped with communication for the next segment (in a manner analogous to Fig. 3, but instead of independent variable we are overlapping segments of the same variable). As for the communication mechanism, the most natural choice is to use nonblocking collectives, namely MPI_Ialltoallv. For comparison, we have also implemented an alternative mechanism, using pairwise exchanges with MPI_Isend and MPI_Irecv nonblocking calls.
3.2 Discussion

In Fig. 5 we show comparison between these implementations on Summit platform, for a range of different \( N_c \) values. We see that the pairwise method performs better in all cases. This result is clearly specific to the given platform and MPI vendor, as well as possibly the problem considered. Thus we are leaving both collective and pairwise communication options in P3DFFT++ for the user to choose.

We also note that there is a sweet spot in performance as a function of number of segments \( N_c \). The same was observed for other grid sizes and core counts. As \( N_c \) increases, so does the degree of overlap \((N_c-1)/N_c\). However, it also decreases the message size, which leads to poor bandwidth utilization, and this starts to affect performance negatively at the higher end of \( N_c \) range.

Figure 5. Comparison of collective communication versus pairwise exchange mechanisms in the all-to-all exchange in nonblocking communication version of P3DFFT++. Results were obtained on Summit at ORNL, using complex 3D FFT of size 2048\(^3\), on 4096 cores. Time of solution is plotted on the vertical axis (less is better).

We compare performance of the overlap version with the default (blocking) version in Figs. 6 and 7. We plot the data from the blocked version, and the overlapped version (pairwise exchange) with \( N_c=1 \), as well as the best \( N_c \) in terms of solution time. The \( N_c=1 \) case implies no overlap is taking place, however it is still interesting to compare against the default case, which uses MPI_Alltoallv.

In many cases we observe improvement with the overlap version, showing up to 22% better performance than the default version. The magnitude of the difference is highest at low core counts, which can be explained by decreasing message size as the core count increases. In some cases the \( N_c=1 \) runs produce better results than \( N_c>1 \) runs. This is likely explained by smaller message sizes, leading to poorer bandwidth utilization and thus cancelling the benefits of the overlap. It is also noteworthy that \( N_c=1 \) runs are typically better than the default version, even though no overlap is taking place. This is likely a reflection of peculiarities of the MPI implementation, with pairwise exchanges providing better performance than MPI_Alltoallv.

Figure 6. Performance comparison between blocking, nonblocking pairwise version without overlap, and nonblocking version with overlap (best \( N_c \)), as a function of number of cores. Time of solution is plotted on the vertical axis (less is better). These results were obtained on Summit supercomputer at ORNL, with complex 3D FFT benchmark test on 1024\(^3\) grid.

Figure 7. Performance comparison between blocking, nonblocking pairwise version without overlap, and nonblocking version with overlap (best \( N_c \)), as a function of number of cores. Time of solution is plotted on the vertical axis (less is better). These results were obtained on Summit supercomputer at ORNL, with complex 3D FFT benchmark test on 2048\(^3\) grid.
Communication Overlap In Large-Scale Spectral Transforms

Although these conclusions may be specific for the system and problem studied, they indicate potential of the overlap approach, as well as the benefit of having a pairwise exchange version, in addition to a collective communication version. The best setting for the Nc parameter could be determined by trial and error for each system/problem combination.

Having the software framework in place for single-variable overlap gives us the possibility to consider further improvements in the algorithm, for example, a threaded implementation, where MPI is combined with OpenMP threads, and each thread is responsible for a segment of the local array.

Finally, this framework will also be used for overlapping GPU-CPU traffic with GPU computation, in an ongoing effort to optimize the GPU version.

4 Conclusions

We have discussed ongoing work on P3DFFT++ open source library for spectral transforms. After reviewing past work and defining the problem, we have focused on different ways of implementing overlap of communication with computation. We have described single-variable overlap implementation and investigated dependence of performance on the number of segments and the mechanism of nonblocking communication. We have shown promising results from benchmarking runs, where we see up to 22\% improvement in performance resulting from overlap with pairwise exchange communication in complex 3D FFT. These results apply, without loss of generality, to other types of spectral transforms besides FFT.

These developments will be part of a public release of P3DFFT++ in the near future. In the future we will also implement multi-variable transforms, where the ability to overlap larger buffers is likely to lead to better performance through bandwidth utilization. We will also study threaded overlap. This work also lays the ground for overlapping GPU traffic with computation, which is an important part of extracting higher performance from GPUs, and will be part of continued work on the package in the near future.

Every effort is made to make the package portable and easy to use, while giving the user maximal controls affecting potential performance of the library.

ACKNOWLEDGMENTS

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

This work was supported by U.S. NSF grant OAC-1835885.

REFERENCES


