Improvements to the Coastal Model Repository (CMR)

by

Christopher Vasquez

Undergraduate Honors Thesis Under the Direction of

Dr. Steve Brandt

Department of Computer Science and Engineering

Submitted to the LSU Roger Hadfield Ogden Honors
College in Partial Fulfillment of
the Upper Division Honors Program

October, 2021

Louisiana State University
& Agricultural and Mechanical College
Baton Rouge, Louisiana

1 Introduction

Allowing for the quick deployment of coastal models, the Coastal Model Repository (CMR) serves as a community-based repository for open-source models used by coastal researchers. Providing a minimalistic interface to interact with HPC machines, the CMR aims to simplify the process of using coastal models for scientific research. Prior to this work, the CMR came installed with the following pre-compiled models: SWAN, NHWave, Funwave, and Openfoam.

The new work documented in this thesis has extended its functionality. Through the integration of Spack and development of a plug-in framework, the CMR now supports custom model installation with improved package management. In addition to adding these features, work has also been done to add Delft3D and Delft3D Flexible Mesh to the pre-installed models list. In adding these new capabilities to the CMR, the project has become more flexible in its user-experience, leaving many possibilities for further development.

2 Science Gateways

A Science Gateway is a user-friendly, community-developed interface that allows access to advanced resources for science and engineering researchers, educators, and students. In wanting to provide user-friendly access to advanced scientific models, the CMR serves as a science gateway for coastal researchers. Allowing all these models to be collectively located in one location, the CMR provides a simplified way to compile, run, monitor job progress, and analyze results. Additionally, with a simple, easily accessibly GUI, anyone can deploy and run a wide set of coastal models with the CMR. With the goal of this project being to facilitate an environment for open-source research and science, the CMR offers maximum flexibility to users, allowing them to utilize the CMR in any way they see fit.

3 Reproducibility and Provenance

A fundamental component of the scientific method, reproducibility is the recreating of previous results to verify their consistency. With regards to software, consistency is potentially dependent on different versions of every element in the software stack as well as the compiler, flags, and other factors. In order to ensure proper reproducibility, software must also have provenance. For instance, if computational results disagree–or a computational result disagrees with a measurement–provenance will be a key aspect ensuring which result, if any, is correct. Furthermore, this fact can help identify the errors and effects in reproducibility when software yields incorrect results. Both of these qualities are critical in scientific software since data measurement must be accurate, precise, and consistent regardless of the software used. In order for computational results to be used to perform experiments using the scientific method, reproducible is essential. However, in order to achieve the expected result, provenance must be present, as it tells exactly what was done and allows others to reproduce results of an experiment. In the context of the CMR, provenance allows users to see the effects of switching compilers, MPI implementations, and other parameters surrounding the execution of models.

With the goal of reproducibility in mind, the CMR aims to deliver provenance through its implementation of an automated package manager, container software and middleware, i.e. Spack, Docker, and Agave (Tapis), respectively. Having the potential to contain a plethora of models, the CMR needed a way to ensure that each of these models and their dependencies are organized and easily accessible for users, especially since each model itself has different versions. Given that some users may prefer to use certain versions of models and dependencies to ensure consistency in their results, the CMR integrated Spack as means of meeting this desire. Spack allows for the automated organization and installation of models, dependencies, and their respective versions so that reproducibility and provence with the models is never an issue. Since Spack stores all information regarding building a model within a single file, users can easily share models with each other to reproduce scientific results, encouraging collaboration among the CMR community.

Similar to Spack, Docker adds an aspect of portability to the CMR which ensures consistency in utilizing the CMR. One of the biggest aspects of the CMR is its usage of Docker containers to maintain a portable client-side environment. With the utilization of Docker, these containers can be pushed to DockerHub and shared with others to review and use their work for scientific collaboration. Since the entire client-side environment is being shared, any worry regarding missing packages in a user's environment becomes irrelevant. Additionally, these containers detail the specific versions of packages installed in the client-side environment, meaning that provenance is maintained within the container. Thanks to the portability of Docker containers via DockerHub, any scientific data generated from the CMR can easily be shared and reproduced by anyone with Docker installed.

In an effort to aid in provenance, Agave (Tapis) localizes files associated with each job in the CMR. Specifically, this middleware packages all input and output files into a single directory currently known as "run_dir." By having these job files organized in this fashion, users can easily analyze and verify the inputs and outputs of their experiments, allowing for themselves and others to easily reproduce the results.

4 Open Source Software

Being open-source, the CMR avoids the troubles of licensing complications and allows greater freedom in conducting science. In contrast with open-source software, proprietary software can often leave users wondering how certain scientific results were generated since the source code is not disclosed. Open-source software naturally allows greater reproducibility since public accessibility allows users to examine and understand how the system works. Additionally, this public accessibility often encourages engagement, often forming communities amongst users. This engagement can lead to the sharing of scientific data, as the project facilitates a place for collaboration. With the CMR being open-source, the project will serve as a source of transparency in scientific experimental methodology, observation, and collection of data.

5 Notebooks

Though using a web page was a possibility during the development of the CMR, making it would have required much more development time. Additionally, the web page would constrain users to the provided interface. Therefore, in wanting to give users control and provide greater flexibility with the CMR, we have chosen to use notebooks in place of a web page.

As mentioned before, notebooks offer much flexibility in terms of usage. Providing generic graphics, notebooks already come with a simple GUI that almost all users can use without the need for any additional graphical packages. In addition to a GUI, notebooks also provide a CLI in the case that the user needs privileged control over the CMR. This CLI availability allows users to fix issues themselves, not needing to wait on the CMR developers for help. Choosing to use notebooks also inherently allows for users to keep all their research in one place while also keep documentation of any past experiments.

With all these listed features an many others not mentioned, notebooks allow users to take control of the project for themselves, giving them freedom in using the CMR.

6 Images

6.1 Docker

Docker [1] is a container software that allows users to run application in a loosely isolated environment. Though many other technologies could have been used in place of Docker, this particular container software is one of the most popular due to its simplicity and low overhead. Since this piece of software is already used by many, the installation process of it has already been made easy, repeatable, and debugged by the developers, making it both reputable and user-friendly. From a more technical standpoint, unlike some other forms of container technologies, Docker leverages the Linux kernel as opposed to creating a full virtual machine, allowing for built images to maintain a smaller size. Furthermore, Dockers' online service 'DockerHub' allows for an easy and convenient means of sharing built images. The combination of these smaller images and the DockerHub service allows for Docker to be one of the most portable technologies in the realm of container software.

In terms of the CMR, the portability of Docker images allows for the CMR environment to easily be shipped around from one machine to another. Additionally, the image itself has been built using X86 to ensure maximum compatibility with almost all machines. Given that all client side and server side code is stored within this docker image, a user needs to only install Docker and pull our image from DockerHub in order to begin using the CMR.

6.2 Singularity

Similar to Docker, Singularity [2] is a piece of container software that allows users to build images that contain loosely isolated environments. The biggest difference between Docker and Singularity, however, is the permissions a user is given when accessing an image. With Docker, users are always given root permissions when stepping inside an image. On the contrary, Singularity allows the builder of an image to control the permissions users have when stepping inside an image. Although Singularity has its own format for images, it can also run docker images, pulling them from the DockerHub and allowing control of permissions Docker does not offer. Given that Singularity is used on the server-side cluster, this small difference is crucial for maintaining the security and functionality of the CMR.

Though there are alternatives to Singularity, such as Charlie Cloud and Shifter, Singularity, like Docker, is the more popular option on them all. Given this fact, clusters are more likely to already have singularity installed than these other alternative technologies, making setting up a cluster for the CMR easier.

6.3 MPI in Images

MPI serves as the standardized way in which machines communicate amongst themselves, usually in a cluster. The CMR uses a cluster as the server primarily for speeding up the time taken to perform intensive calculations done by models within it. Therefore, the need for MPI for performing the computations in parallel is a must for the CMR. In terms of the implementation, MPICH [3] was chosen as the default for MPI on singularity images located on the HPC machines. Although OpenMPI [4] is also another popular choice for MPI, MPICH's increased reliability with specifying a way for an "ssh" executable made it the better choice of the two. However, users still have the option to change the default to OpenMPI or any other form of MPI via Spack.

7 Agave (Tapis)

Agave (Tapis) [5] is an "open platform-as-a-service built by TACC designed for scalable and reproducible research that integrates HPC resources which can be managed through a single API." [5] This service, put simply, manages and run jobs for the users on HPC machines. Particularly, this service allows machines to be setup with a login feature in order to allow multiple users to access the same machine and run the CMR application. Through this technology, the CMR is able to easily allow multiple users to use any given machine and submit jobs while monitoring their progress.

7.1 JetLag

Jetlag [6] serves as interface code that makes interacting with Agave (Tapis) easier by providing a simple GUI for users to interact with. Since simplicity and easy-accessibility are two of the main goals of the CMR, Jetlag's simplification of Agave (Tapis) for users helps to not only reach these goals but also saves time and effort in not requiring users to learn the command line tools of Agave (Tapis).

8 Spack

Spack [7] allows users to build packages with multiple versions, configurations, platforms, and compilers. These builds can easily be generated using a simple Python script which Spack deems as the "package.py file." This file contains the build and install instructions in addition to other information such as package versions and dependencies. Since the author of the "package.py file" writes it in a generic way, the deployer of the file can choose the dependencies. With Spack providing boilerplate for this script file, building custom packages can be done in a matter of minutes. Had Spack not been implemented into the CMR, installing one model could take hours since dependencies need to be installed in a specific order and the versions of these dependencies matter according the the model version installed. In terms of the CMR, this hassle-free building method allows for users to easily add their own models to the CMR.

In addition to providing flexibility and easy-accessibility with package building, Spack allows multiple versions of the same packages to coexist on the same machine without any conflict. Since Spack installs every unique package and dependency configuration into its own prefix, new installs will not break existing ones. This fact allows the CMR to easily keep all its models and their respective versions in one place in an organized manner.

Currently, the CMR is organized in a manner where Spack is present on the HPC machines' singularity images and the HPC machines themselves. Spack's being present on both of these platforms is necessary in order to allow the installation of new models. Since the singularity image is read-only, models need to be installed on the HPC Spack installation (refer to section 6 regarding images). Due to a feature known as chaining, the HPC Spack installation can share all its builds with the Singularity-image Spack installation, allowing for the CMR to utilize any user installed models.

9 Development of the CMR for Thesis

9.1 Spack Integration

Prior to the work conducted for this thesis, if a new model was to be installed on the CMR, users would need to edit menus on the client-side and scripts on the server-side, making the process rather complicated. Since a poor version of Spack had been implemented using bash scripts, the CMR did not provide a dynamic list of the installed packages, meaning that much of the hard-coded package lists was still riddled throughout the project. This lack of automation in the process often led to less flexibility since packages needed to be installed according to the required dependencies. Therefore, in wanting to resolve this issue, work was conducted in order to integrate Spack, an automated package manager, into the CMR. This new addition to the CMR would solve the previously mentioned problems and lead to greater flexibility in expanding the capabilities of the CMR.

When adding this technology to the CMR, much of the original source code on both the client and server sides needed to be modified. In particular, with regards to the server-side, the Spack package manager needed be installed on both a singularity image on the HPC machine and the user's home directory on the HPC machine itself. Given that the singularity image is read-only, in order for models to be installed, the spack installation in the user's home directory will be linked to that of the singularity image, allowing packages to be installed in the user's home directory yet still readable by the singularity image.

With Spack setup on the server-side, the client-side now needed to be configured to read the packages installed on Spack and be able to execute them accordingly. In order to accomplish the former, a simple bash script was passed through a job to the HPC machine which then returned a text file containing a list of all the models installed on the machine. Since this process is relatively slow (around 3 minutes), this process only occurs once when the user first installs the CMR. If new models are ever installed on the CMR, the user can always update the list of installed models by clicking an update button which will repeat the same initially mentioned process.

Before the addition to Spack on the CMR, executing a model on the HPC machine required the client to pass a specific shell script to the HPC machine, resulting in a plethora of script files for designated for each model. Since Spack automatically calls all dependencies when running a model, a generic script file was able to be written for any model's execution on any HPC machine. This script file would still be passed to the HPC machine, but ensure that only one file will ever be necessary to run any given number of models on the CMR.

9.2 Plug-In Framework & Delft-3D

With Spack simplifying the the process of installing packages, the CMR can now provide users with a simple process of installing custom models. In wanting to expand the models offered by the CMR, work was conducted to allow any type of model to be installed. This plug-in framework was made possible due to Spack's ability to build models using only a python file. This script file details the dependencies, versions, and any building instructions, allowing Spack to know how to build a given model. By taking advantage of this feature of Spack, a plug-in framework was able to be developed for the installation of user's custom models.

Being strictly server-side, the plug-in framework needs to be setup via a script file and configured by the user to allow custom models. Once these steps are complete, the user simply needs to construct a tarball containing the "package.py file" for his/her given model, a JSON file detailing specifications regarding his model (See Figure 1), and a tarball of the source code of the model. Once this tarball is provided to the plug-in framework, it will install the model onto the machine, allowing the client that connects to that machine to now access that newly installed custom model. Since users are able to add models themselves, they are also able to add proprietary models so long as they follow the same process for adding a model to the CMR.

```json
{
    "name": "SWAN",
    "package": "swan",
    "version": "4.1.3.1",
    "inputFile": "INPUT",
    "nx": "PX",
    "ny": "PY",
    "nz": ""
}
```

Figure 1: JSON File Format

In testing the plug-in framework, Delft3D served as the first model to be successfully installed. Using the method described above, Delft3D was successfully built and is now included with the CMR as a pre-installed model.

9.3 Notebook Improvements

In integrating Spack and developing a plug-in framework, multiple changes were made to the model notebook, which is responsible for providing the interface for interacting with the various models and submitting jobs. Initially, before the work mentioned in this thesis was conducted on the CMR, the project contained many nonworking models and features such as output text boxes and labels. While overhauling the package management of the CMR, the notebook interface was adapted to fit the needs of the new CMR features. As expected, certain features became deprecated and were removed while others were completely changed or restored to their working states.

In addition to the main model notebook, however, the SWAN visualization notebook also received a major changes in order to restore it to a working state. This notebook is responsible for allowing the outputs of the SWAN model to be visualized through videos and graphs (See Figure 2). This particular work was done in conjuntion with Said Lababidi, a former student of Northeastern University in Boston, MA.

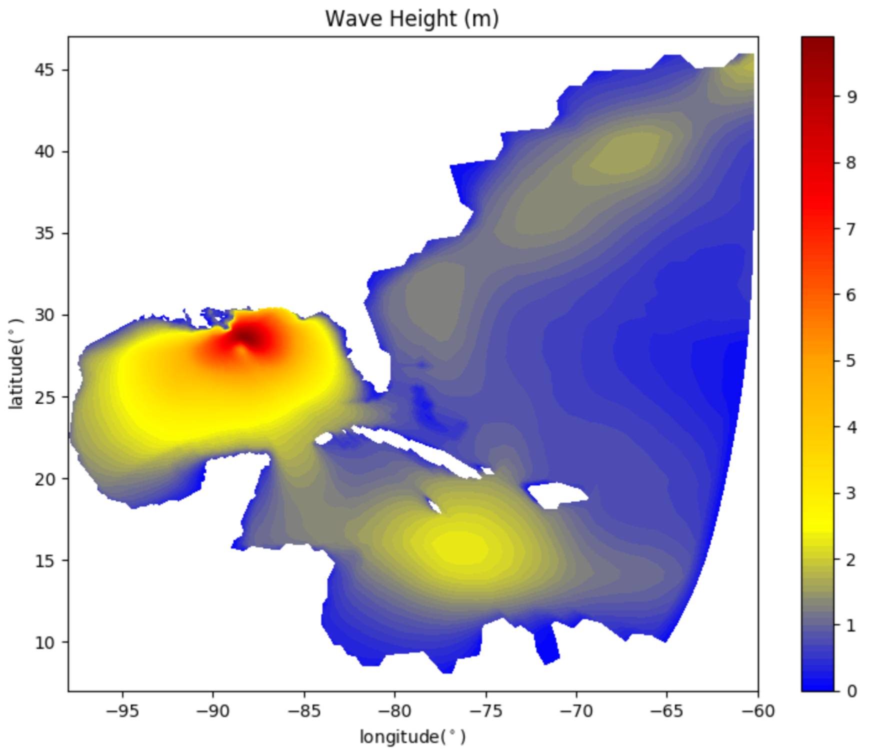

Figure 2: 2D Wave Model Generated from SWAN Visualization Notebook

9.4 CMR Wiki & Documentation

With various new features added to the CMR, the operation of the project needed to be documented in order for users to properly utilize it. Therefore, a wiki was created in order to document all updates and instructions for the CMR. The wiki is currently still in early development, but this page will act as the main form of documentation for the CMR.

10 Coastal Science Models

10.1 SWAN

SWAN [8] is one of the 4 original models included within the CMR. "As a third-generation wave model, SWAN computes random, short-crested wind-generated waves in coastal regions and inland waters. In particular, the model accounts for the following physics: Wave propagation in time and space, shoaling, refraction due to current and depth, frequency shifting due to currents and non-stationary depth, Wave generation by wind, Three- and four-wave interactions, Whitecapping, bottom friction and depth-induced breaking, Dissipation due to vegetation, Wave-induced set-up, Propagation from laboratory up to global scales, Transmission through and reflection (specular and diffuse) against obstacles, and Diffraction." [8]

10.2 OPENFOAM

OpenFOAM [9], like SWAN, is a model that has always been included in the CMR. "The model is a free, open source computational fluid dynamics (CFD) software package released by the OpenFOAM Foundation. It has a large user base across most areas of engineering and science from both commercial and academic organizations. This model has an extensive range of features to solve anything from complex fluid flows involving chemical reactions, turbulence and heat transfer, to solid dynamics and electromagnetics." [9]

10.3 NHWAVE

NHWAVE [10] is another model that was included with the first version of the CMR. "This model is a three-dimensional shock-capturing Non-Hydrostatic WAVE model which solves the incompressible Navier-Stokes equations in terrain and surface-following sigma coordinates. The model predicts instantaneous surface elevation and 3D flow field and is capable of resolving coastal wave processes (shoaling, refraction, diffraction, breaking etc.) as well as tsunami wave generation by submarine mass failure. The governing equations are discretized by a shock-capturing Godunov-type numerical scheme. A nonlinear Strong Stability-Preserving (SSP) Runge-Kutta scheme is adopted for adaptive time stepping with second-order temporal accuracy. The model is fully parallelized using Message Passing Interface (MPI) with non-blocking communication. The poisson equation is solved by the high performance preconditioner HYPRE software library." [10]

10.4 FUNWAVE-TVD

FUNWAVE–TVD [11] is the last model that was included with the original CMR. "This particular model is the Total Variation Diminishing (TVD) version of the fully nonlinear Boussinesq wave model (FUNWAVE) developed. The development of the present version was motivated by recent needs for modeling of surfzone–scale optical properties in a Boussinesq model framework, and modeling of Tsunami waves in both a global/coastal scale for prediction of coastal inundation and a basin scale for wave propagation." [11]

10.5 DELFT3D

Delft3D [12] is one of the two new models added to the pre-installed models in the CMR. "It is an integrated modelling suite, which simulates two-dimensional (in either the horizontal or a vertical plane) and three-dimensional flow, sediment transport and morphology, waves, water quality and ecology and is capable of handling the interactions between these processes. The suite is designed for use by domain experts and non-experts alike, which may range from consultants and engineers or contractors, to regulators and government officials, all of whom are active in one or more of the stages of the design, implementation and management cycle." [12]

10.6 DELFT3D-FM

The Delft3D Flexible Mesh Suite [12] (Delft3D FM) is another recent addition to the CMR's list of pre-installed models. "It is the successor of the structured Delft3D 4 Suite. Like Delft3D 4, the Delft3D FM Suite can simulate storm surges, hurricanes, tsunamis, detailed flows and water levels, waves, sediment transport and morphology, water quality and ecology, and is capable of handling the interactions between these processes. The suite is designed for use by domain experts and non-experts alike, which may range from consultants and engineers or contractors, to regulators and government officials, all of whom are active in one or more of the stages of the design, implementation and management cycle. cycle." [12]

11 Future Work

11.1 Addition of More Models

With the plug-in framework offering the capability of installing custom models, the CMR is now able to easily expand its library of coastal models. In wanting to offer more pre-installed coastal models, work is being conducted to implement the models XBeach [13] and Cactus [14] into the CMR as pre-installed models in the near future. Additionally, experiments in adding proprietary models to the CMR will also be researched; though closed-source software will not be able to be added as pre-installed models, allowing users to have this option can greatly increase the appeal of this project in various aspects of research.

11.2 Running Models on the Client

Currently, the CMR is fully dependent upon remote HPC machines to run jobs with models, requiring users to always use them. Given that HPC machines can handle any amount of intense computation, having the CMR rely on these machines ensures that computational performance is never an issue, especially for large jobs. However, smaller jobs do not require a demanding amount of resources and, thus, do not need to be run on an HPC machine. Therefore, work is currently being conducted to develop an option for users to run models on their own local machine. Similar to allowing users to change their implementation of MPI, users will be able to change where they run their jobs, giving greater flexibility to the users.

11.3 API Description Language Integration

In an effort to further increase the flexibility of the CMR, the usage of an API description language in the project will allow it to become usable by other software applications. With this project being open-source, allowing users to integrate the CMR into their frameworks via this new technology can potentially add more users to the CMR. Additionally, API description languages also double as a form of documentation, being both human-readable and machine-readable. Research still needs to be conducted on this topic, but its capabilities are promising for bettering the CMR's use-cases among users.

11.4 Feedback & Improvements

As of the time writing, the CMR is being re-introduced to students at Northeastern University in Boston, Massachusetts. In wanting the better the user experience of the CMR, any feedback provided from these students or other users will be taken into account as further development is conducted on the project.

11.5 Presentation & Publication

As mentioned prior, one of the goals of the CMR is to serve as a science gateway, allowing coastal researches and educators alike to utilize scientific models. Within the next year, the work on the CMR mentioned in this thesis is planned for presentation at a gateways conference, such as one hosted by the International Workshop on Science Gateways (IWSG). The paper written and presented at this conference will also be looked into being published in the future.

12 Conclusion

Though the previous version of the CMR offered multiple coastal models in a user-friendly environment, it failed to deliver in its ability to expand its library of models in a simple fashion. Lacking modularity, the CMR needed a solution to overcome this major flaw.

Spack served as the main solution to many of the CMR's problems. By integrating this technology into the project, package management was made simple and the potential for expansion grew. From this potential came the development of the plug-in framework for easy model installation which fully realized the idea of the CMR becoming a repository of coastal models. Through the implementation Spack and introduction of a plug-in framework, the CMR became a new tool not only for coastal models but any type of models.

As development on the CMR continues, the project will continue to grow by implementing new models with the plug-in framework and increasing the user experience for scientist and researchers, fulfilling the goal of a becoming a true science gateway.

13 Acknowledgements

This work was supported by NSF award CCF 1539567 [15].

References

[1] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.

[2] Gregory M. Kurtzer, Vanessa Sochat, and Michael W. Bauer. Singularity: Scientific containers for mobility of compute. *PLOS ONE*, 12(5):1–20, 05 2017.

[3] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. A high-performance, portable implementation of the mpi message passing interface standard. *Parallel Computing*, 22(6):789–828, 1996.

[4] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall. Open mpi: Goals, concept, and design of a next generation mpi implementation. In Dieter Kranzlmüller, Péter Kacsuk, and Jack Dongarra, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 97–104, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[5] Sean B Cleveland, Anagha Jamthe, Smruti Padhy, Joe Stubbs, Steven Terry, Julia Looney, Richard Cardone, Michael Packard, Maytal Dahan, and Gwen A Jacobs. Tapis v3 streams api: Time-series and data-driven event support in science gateway infrastructure. *Concurrency and Computation: Practice and Experience*, 33(19):e6103, 2021.

[6] Steven R. Brandt, Alex Bigelow, Sayef Azad Sakin, Katy Williams, Katherine E. Isaacs, Kevin Huck, Rod Tohid, Bibek Wagle, Shahrzad Shirzad, and Hartmut Kaiser. Jetlag: An interactive, asynchronous array computing environment. In *Practice and Experience in Advanced Research Computing*, PEARC '20, page 8–12, New York, NY, USA, 2020. Association for Computing Machinery.

[7] Todd Gamblin, Matthew LeGendre, Michael R. Collette, Gregory L. Lee, Adam Moody, Bronis R. de Supinski, and Scott Futral. The spack package manager: Bringing order to hpc software chaos. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, New York, NY, USA, 2015. Association for Computing Machinery.

[8] N Booij, LH Holthuijsen, and RC Ris. The" swan" wave model for shallow water. In *Coastal Engineering 1996*, pages 668–676. 1997.

[9] Niels G Jacobsen, David R Fuhrman, and Jørgen Fredsøe. A wave generation toolbox for the open-source cfd library: Openfoam®. *International Journal for numerical methods in fluids*, 70(9):1073–1088, 2012.

[10] Morteza Derakhti, James T Kirby, Fengyan Shi, and Gangfeng Ma. Nhwave: Consistent boundary conditions and turbulence modeling. *Ocean Modelling*, 106:121–130, 2016.

[11] James T Kirby, Ge Wei, Qin Chen, Andrew B Kennedy, and Robert A Dalrymple. Funwave 1.0: fully nonlinear boussinesq wave model-documentation and user's manual. *research report NO. CACR-98-06*, 1998.

[12] JA Roelvink and GKFM Van Banning. Design and development of delft3d and application to coastal morphodynamics. *Oceanographic Literature Review*, 11(42):925, 1995.

[13] Dano Roelvink, AJHM Reniers, A Van Dongeren, J Van Thiel de Vries, Jamie Lescinski, and Robert McCall. Xbeach model description and manual. *Unesco-IHE Institute for Water Education, Deltares and Delft University of Tecnhology. Report June*, 21:2010, 2010.

[14] Frank Löffler, Steven R Brandt, Gabrielle Allen, and Erik Schnetter. Cactus: Issues for sustainable simulation software. *arXiv preprint arXiv:1309.1812*, 2013.

[15] Cybersees: Type 2: A coastal resilience collaboratory: Cyber-enabled discoveries for sustainable deltaic coasts, 2015.