Clawpack: building an open source ecosystem for solving hyperbolic PDEs

Kyle T. Mandli* Aron J. Ahmadia† Marsha J. Berger‡ Donna A. Calhoun§
David L. George¶ Yiannis Hadjimichael‖ David I. Ketcheson**
Grady I. Lemoine†† Randall J. LeVeque‡‡

May 13, 2016

Abstract

Clawpack is a software package designed to solve nonlinear hyperbolic partial differential equations using high-resolution finite volume methods based on Riemann solvers and limiters. The package includes a number of variants aimed at different applications and user communities. Clawpack has been actively developed as an open source project for over 20 years. The latest major release, Clawpack 5, introduces a number of new features and changes to the code base and a new development model based on GitHub and Git submodules. This article provides a summary of the most significant changes, the rationale behind some of these changes, and a description of our current development model.

1 Introduction

The Clawpack software suite [14] is designed for the solution of nonlinear conservation laws, balance laws, and other first-order hyperbolic partial differential equations not necessarily in conservation form. The underlying solvers are based on the wave propagation algorithms described by LeVeque in [39], and are designed for logically Cartesian uniform or mapped grids or an adaptive hierarchy of such grids. The original Clawpack was first released as a software package in 1994 and since then has made major strides in both capability and interface. More recently a major refactoring of the code and a move to GitHub for development has resulted in the release of Clawpack 5.0 in January, 2014. Beyond enabling a distributed and better managed development process a number of user-facing improvements were made including a new user interface and visualization tools, incorporation of high-order accurate algorithms, parallelization through MPI and OpenMP, and other enhancements.

Because scientific software has become central to many advances made in science, engineering, resource management, natural hazards modeling and other fields, it is increasingly important to describe and document changes made to widely used packages. Such documentation efforts serve to orient new and existing users to the strategies taken by developers of the software, place the software package in

*Columbia University (kyle.mandli@columbia.edu)
†Continuum Analytics (aahmadia@continuum.io)
‡New York University (berger@cims.nyu.edu)
§Boise State University (donnacalhoun@boisestate.edu)
¶USGS Cascades Volcano Observatory (dgeorge@usgs.gov)
‖King Abdullah University of Science and Technology, Box 4700, Thuwal, Saudi Arabia, 23955-6900 (yiannis.hadjimichael@kaust.edu.sa)
**King Abdullah University of Science and Technology, Box 4700, Thuwal, Saudi Arabia, 23955-6900 (david.ketcheson@kaust.edu.sa)
††CD-Adapco, Bellevue, WA
‡‡Dept. of Applied Mathematics, University of Washington (rjl.@uw.edu)

the context of other packages, document major code changes, and provide a concrete, citable reference for users of the software.

With this in mind, the goals of this paper are to:

- Summarize the development history of Clawpack,
- Summarize some of the major changes made between the early Clawpack 4.x versions and the most recent version, Clawpack 5.3,
- Summarize the development model we have adopted, for managing open source scientific software projects with many contributors, and
- Identify how users can contribute to the Clawpack suite of tools.

This paper provides a brief history of Clawpack in Section 1.1, a background of the mathematical concerns in Section 1.2, the modern development approach now being used in Section 2, the major feature additions in the Clawpack 5.x major release up until Version 5.3 in Section 3. Some concluding thoughts and future plans for Clawpack are mentioned in Section 4.

1.1 History of Clawpack

The first version of Clawpack was released by LeVeque in 1994 [37] and consisted of Fortran code for solving problems on a single, uniform Cartesian grid in one or two space dimensions, together with some Matlab [45] scripts for plotting solutions. The wave-propagation method implemented in this code provided a general way to apply recently developed high-resolution shock capturing methods to general hyperbolic systems and required only that the user provide a "Riemann solver" to specify a new hyperbolic problem. Collaboration with Berger [9] soon led to the incorporation of adaptive mesh refinement (AMR) in two space dimensions, and work with Langseth [36, 35] led to three-dimensional versions of the wave-propagation algorithm and the software, with three-dimensional AMR then added by Berger.

Version 4.3 of Clawpack contained a number of other improvements to the code and formed the basis for the examples presented in a textbook [39] published in 2003. That text not only provided a complete description of the wave propagation algorithm, developed by LeVeque, but also is notable in that the codes used to produce virtually all of figures in the text were made available online [39].

In 2009, Clawpack Version 4.4 was released with a major change from Matlab to Python as the recommended visualization tool, and the development of a Python user interface for specifying the input data.

In 2009, Clawpack Version 4.4 was released with a major change from Matlab to Python as the recommended visualization tool, and the development of a Python user interface for specifying the input data. Finally in January of 2013 the 4.x versions of Clawpack ended with the release of 4.6.3[1]

Version 5 of Clawpack introduces both user-exposed features and a number of modern approaches to code development, interfacing with other codes, and adding new capabilities. The move to `git` version control also allowed a more complete open source model. These changes are the subject of the rest of this paper.

1.2 Hyperbolic problems

In one space dimension, the hyperbolic systems solved with Clawpack typically take the form of conservation laws

$$q_t(x,t) + f(q(x,t))_x = 0 \tag{1}$$

[1] Details of these changes can be found at `http://depts.washington.edu/clawpack/users-4.6/changes.html`. Version 4.x used `svn` version control and the freely available software (under the BSD license) was distributed via tarballs.

or non-conservative linear systems

$$q_t(x,t) + A(x)q(x,t)_x = 0, \tag{2}$$

where subscripts denote partial derivatives and $q(x,t)$ is a vector with $m \geq 1$ components. Here the components of q represent conserved quantities, while the function f represents the flux (transport) of q. Equation (1) generalizes in a natural way to higher space dimensions; see the examples below. The coefficient matrix A in (2) or the Jacobian matrix $f'(q)$ in (1) is assumed to be diagonalizable with real eigenvalues for all relevant values of q, x, and t. This condition guarantees that the system is hyperbolic, with solutions that are wave-like. The eigenvectors of the system determine the relation between the different components of the system, or waves, and the eigenvalues determine the speeds at which these waves travel. The right hand side of these equations could be replaced by a "source term" $\psi(q,x,t)$ to give a non-homogeneous equation that is sometimes called a "balance law" rather than a conservation law. Spatially-varying flux functions $f(q,x)$ in (1) can also be handled using the f-wave approach [5].

Examples of equations solved by Clawpack include:

- Advection equation(s) for one or more tracers; in the simplest, one-dimensional case we have:

$$q_t + (u(x,t)q)_x = 0.$$

  The velocity field $u(x,t)$ is typically prescribed from the solution to another fluid flow problem, such as wind. Typical applications include transport of heat, energy, pollution, smoke, or another passively-advected quantity that does not influence the velocity field.

- The shallow water equations, describing the velocity (u,v) and surface height h of a fluid whose depth is small relative to typical wavelengths.

$$h_t + (hu)_x + (hv)_y = 0 \tag{3}$$

$$(hu)_t + \left(hu^2 + \frac{1}{2}gh^2\right)_x + (huv)_y = -gb_x \tag{4}$$

$$(hv)_t + \left(hv^2 + \frac{1}{2}gh^2\right)_y + (huv)_x = -gb_y \tag{5}$$

$$\tag{6}$$

  Here g is a constant related to the gravitational force and $b(x,y)$ is the *bathymetry*, or bottom surface height. Notice that the bathymetry enters the equations through a *source term*; additional terms could be added to model the effect of bottom friction. These equations are used, for instance, to model inundation caused by tsunamis and dam breaks, as well as to model atmospheric flows.

- The Euler equations of compressible, inviscid fluid dynamics, consist of conservation laws for mass, momentum, and energy. The wave speeds depend on the local fluid velocity and the acoustic wave velocity (sound speed). Source terms can be added to include the effect of gravity, viscosity or heat transfer. These systems have important applications in aerodynamics, climate and weather modeling, and astrophysics.

- Elastic wave equations, used to model compressional and shear waves in solid materials. Here even linear models can be complex due to varying material properties on multiple scales that affect the wave speeds and eigenvectors.

Discontinuities (shock waves) can arise in the solution of nonlinear hyperbolic equations, causing difficulties for traditional numerical methods based on discretizing derivatives directly. Modern shock capturing methods are often based on solutions to the *Riemann problem* that consists of equations (1) or (2) together with piecewise constant initial data with a single jump discontinuity. The solution to the Riemann problem is a similarity solution (a function of x/t only), typically consisting of m waves (for a system of m equations) propagating at constant speed. This is true even for nonlinear problems, where the waves may be shocks or rarefaction waves (through which the solution varies continuously in a self-similar manner).

The main theoretical and numerical difficulties of hyperbolic problems involve the prescription of physically correct weak solutions and understanding the behavior of the solution at discontinuities. The Riemann solver is an algorithm that encodes the specifics of the hyperbolic system to be solved, and it is the only routine (other than problem-specific setup such as initial conditions) that needs to be changed in order to apply the code to different hyperbolic systems. In some cases, the Riemann solver may also be designed to enforce physical properties like positivity (e.g., for the water depth in GeoClaw) or to account for forces (like that of gravity) that may be balanced by flux terms.

Clawpack is based on Godunov-type finite volume methods in which the solution is represented by cell averages. Riemann problems between the cell averages in neighboring states are used as the fundamental building block of the algorithm. The wave-propagation algorithm originally implemented in Clawpack (and still used in much of the code) is based on using the waves resulting from each Riemann solution together with limiter functions to achieve second-order accuracy where the solution is smooth together with sharp resolution of discontinuities without spurious numerical oscillations (see [39] for a detailed description of the algorithms). Higher-order WENO methods have also been developed relying on the same Riemann solvers. These methods can be found in PyClaw (see Section 3.6), one of the packages in the larger Clawpack ecosystem.

Problem-specific boundary conditions must also be imposed, which are implemented by a subroutine that sets the solution value in *ghost cells* exterior to the domain each time step. The Clawpack software contains library routines that implement several sets of boundary conditions that are commonly used, *e.g.* periodic boundary conditions, reflecting solid wall boundary conditions for problems such as acoustics, Euler, or shallow water equations, and non-reflecting (absorbing) extrapolation boundary conditions. As with all Clawpack library routines, the boundary condition routine can be copied and modified by the user to implement other boundary conditions needed for a particular application.

In two or three space dimensions, the wave-propagation methods are extended using either dimensional splitting, so that only one-dimensional Riemann solvers are needed, or by a multi-dimensional algorithm based on *transverse Riemann solvers* introduced in [38]. Both approaches are supported in Clawpack. A variety of Riemann solvers have been developed for Clawpack, many of which are collected in the `riemann` repository, see Section 3.2.

Adaptive mesh refinement (AMR) is essential for many problems and has been available in two space dimensions since 1995, when Marsha Berger joined the project team and her AMR code for the Euler equations of compressible flow was generalized to fit into the software which became AMRClaw [10], another package included in the Clawpack ecosystem. AMRClaw was carried over to three space dimensions using the unsplit algorithms introduced in [36]. Starting in Version 5.3.0, dimensional splitting is also supported in AMRClaw, which can be particularly useful in three space dimensions where the unsplit algorithms are much more expensive. Other recent improvements to AMRClaw are discussed in Section 3.4.

There are several other open source software projects that provide adaptive mesh refinement for hyperbolic PDEs. The interested reader may want to investigate AMROC [16], BoxLib[2], Chombo [1], Gerris [50], OpenFOAM [46], or SAMRAI [3], for example.

2 Development Approach

Clawpack's development model is driven by the needs of its developer community. The Clawpack project consists of several interdependent projects: core solver functionality, a visualization suite, a general adaptive mesh refinement code, a specialized geophysical flow code, and a massively parallel Python framework. Changes to the core solvers and visualization suite have a downstream effect on the

[2] https://ccse.lbl.gov/BoxLib/index.html

other codes, and the developers largely work in an independent, asynchronous manner across continents and time zones.

The core CLAWPACK software repositories are:

- `clawpack` – responsible for installation and coordination of other repositories,
- `riemann` – Riemann solvers used by all the other projects,
- `visclaw` – a visualization suite used by all the other projects,
- `clawutil` – utility functions used by most other projects,
- `classic` – the original single grid methods in 1, 2, and 3 space dimensions,
- `amrclaw` – the general adaptive mesh refinement framework in 2 and 3 dimensions,
- `geoclaw` – solvers for depth-averaged geophysical flows which employs the framework in `amrclaw`, and
- `pyclaw` – a Python implementation and interface to the CLAWPACK algorithms including high-order methods and massively parallel capabilities.

A release of CLAWPACK downloaded by users contains all of the above. The repositories `riemann`, `visclaw`, and `clawutil` are sometimes referred to as *upstream* projects, since their changes affect all the remaining projects in the above list, commonly referred to as *downstream* projects. There are some variations on this, for instance AMRCLAW is upstream of GEOCLAW, which uses many of the algorithms and software base from AMRCLAW. To coordinate this the `clawpack` repository points to the most recent known-compatible version of each repository.

Beyond the major core code repositories, additional repositories contain documentation and extended examples for using the packages:

- `doc` – the primary documentation source files. These files are written in the markup language reStructured Text[3], and are then converted to html files using Sphinx[4]. Other documentation such as drafts of this paper are also found in this repository.
- `clawpack.github.com` – the html files created by Sphinx in the `doc` repository are pushed to this repository, and are then automatically served on the web. These appear at http://www.clawpack.org, which is configured to point to http://clawpack.github.com. The name of this repository follows GitHub convention for use with GitHub Pages[5].
- `apps` – applications contributed by developers and users that go beyond the introductory examples included in the core repositories.

The CLAWPACK 4.x code is also available in the repository `clawpack-4.x` but is no longer under development.

2.1 Version Control

The CLAWPACK team uses the Git distributed version control system to coordinate development of each major project. The repositories are publicly coordinated under the CLAWPACK organization on GitHub[6] with the top-level `clawpack` super-repository responsible for hosting build and installation tools, as well as providing a synchronization point for the other repositories. The remaining "core CLAWPACK repositories" listed above are subrepositories of the main `clawpack` organization.

[3] http://www.sphinx-doc.org/en/stable/rest.html
[4] http://sphinx-doc.org
[5] https://pages.github.com/
[6] https://github.com/clawpack

GitHub itself is a free provider of public Git repositories. In addition to repository hosting, the Clawpack team uses GitHub for issue tracking, code review, automated continuous integration via Travis CI[7], and test coverage tracking via Coveralls[8] for the Python-based modules. The issue tracker on GitHub supports cross-repository references, simplifying communication between Clawpack developer sub-teams. The Travis CI service, which provides free continuous integration for publicly developed repositories on GitHub, runs Clawpack's test suites through `nose`[9] on proposed changes to the code base, and through a connection to the Coveralls service, reports on any test failures as well as changes to test coverage.

2.2 Submodules

The `clawpack` "super-repository" serves two purposes. First, it contains installation utilities for each of the sub-projects. Second, it serves as a synchronization point for the project repositories. The remainder of this section provides more details on how Git submodules enable this synchronization.

Whenever possible, teams of software developers coordinate their development in a single unified repository. In situations where this isn't possible, one option provided by Git is the submodule, which allows a super-repository (in this case, `clawpack`), to nest sub-repositories as directories, with the ability to capture changes to sub-repository revisions as new revisions in the super-repository. Under the hood, the super-repository maintains pointers to the location of each submodule and its current revision. The submodule directories contain normal Git repositories, all of the coordination happens in the super-repository.

Each of the other core Clawpack repositories listed above is a submodule of the `clawpack` repository. Every commit that creates a new revision to the `clawpack` repository describes top-level installation code as well as the revisions of each of the submodules. In this way, Git submodules allow Clawpack team members to work asynchronously on independent projects while reusing and maintaining common software infrastructure.

Typically the Clawpack developers advance the master development branch of the top-level `clawpack` repository any time a major feature is added or a bug is fixed in one of the upstream projects that might affect code in other repositories. By checking out a particular revision in the `clawpack` repository and performing a `git submodule update`, all repositories can be updated to versions that are intended to be consistent and functional.

In particular, when Travis CI runs the regression tests in any project repository (performed automatically for any pull request), it starts by installing Clawpack on a virtual machine and the current head of the `clawpack/master` branch indicates the commit from each of the other projects that must be checked out before performing the tests. If the `clawpack` repository has not been properly updated following changes in other upstream projects, these tests may fail.

Any new release of Clawpack is a snapshot of one particular revision of `clawpack` and the related revisions of all submodules. These particular revisions are also tagged for future reference with consistent names, such as `v5.3.1`. (Git tags simply provide a descriptive name for a particular revision rather than having to refer to a Git hash code.)

2.3 Contributing

Scientists who program are often discouraged from sharing code due to existing reward mechanisms and the fear of being "scooped". However, recent studies indicate that scientific communities that openly share and develop code have an advantage because each researcher can leverage the work of many others [53], and that paper citation rates can be increased by sharing code [54] and/or data [49]. Moreover, journals and funding agencies are increasingly requiring investigators to share code used to obtain published results. One of the goals of the Clawpack project is to facilitate code sharing by

[7] https://travis-ci.org/

[8] http://coveralls.io

[9] https://nose.readthedocs.org

users, by providing an easy mechanism to refer to a specific version of the Clawpack software and ensuring that past versions of the software remain available on a stable and citable platform.

On the development side, we expect that the open source development model with important discussions conducted in public will lead to further growth of the developer community and additional contributions from users. Over the past twenty years, many users have written code extending Clawpack with new Riemann solvers, algorithms, or domain-specific problem tools. Unfortunately, much of this code did not make it back into the core software for others to use. Many of the development changes in Clawpack 5.x were done to encourage contributions from a broader community. We have begun to see an increase in contributions from outside the developers' groups, and hope to encourage more of this in the future.

The primary development model is typical for GitHub projects: a contributor forks the repository on GitHub, then develops improvements in a branch that is pushed to her own fork. She issues a "pull request" (PR) when the branch is ready to be merged into the main repository. Increasingly, contributors are also using PRs as a way to conveniently post preliminary or prototype code for discussion prior to further development, often marked WIP for "work in progress" to signal that it is not ready to merge.

After a PR is issued, other developers, including one or more of the maintainers for the corresponding project, review the code. The Travis CI server also automatically runs the tests on the proposed new code. The test results are visible on the GitHub page for the PR. Usually there is some iteration as developers suggest improvements or discuss implementation choices in the code. Once the tests are passing and it is agreed that the code is acceptable, a maintainer merges it.

An additional benefit of using the GitHub platform is that any version of the code is accessible either through the command line `git` interface, through the GitHub website, or a number of available applications on all widely used platforms. More important however is the ability to tag a particular version of a repository with a digital object identifier (DOI) via GitHub and Zenodo[10]. The combination of these abilities provides the capability for Clawpack to not only be accessible at any version but also allows for the citability of versions of the code used for particular results within the scientific literature.

2.4 Releases

Although Clawpack is continuously developed, it is convenient for users to be able to install stable versions of the software. The Clawpack developers provide these releases through two distribution channels: GitHub and the Python Package Index (PyPI). Full source releases are available on GitHub. Alternatively, the PyClaw subproject and its dependencies can be installed automatically using a PyPI client such as `pip`.

Clawpack does not follow a calendar release cycle. Instead, releases emerge when the developer community feels enough changes have accumulated since the last release to justify the cost of switching to a new release. For the most part, Clawpack releases are versioned using an $M.m.p$ triplet, representing the major (M), minor (m), and patch (p) versions respectively. In the broader software engineering community, this is often referred to as semantic versioning. Small changes that fix bugs and cosmetic issues result in increments to the patch-level. Backwards-compatible changes result in an increase to the minor version. The introduction of backwards-incompatible changes require that the major version be incremented. In addition, the implementation of significant new algorithms or capability will also justify the increment of major release number, and is often an impetus for providing another release to the public. In practice, the Clawpack software has frequently included changes in minor version releases that were not entirely backwards compatible, but these have been relatively minor and documented in the release notes. Major version numbers have changed infrequently and related to major refactoring of the code as in going from 4.x to 5.0.

Staring with Version 5.3.1, the tarfiles for Clawpack releases will also be archived on Zenodo[11], a data repository hosted at CERN that issues DOIs so that the software version can be cited with a

[10] For a guide on creating a DOI to a particular version of software see `http://guides.github.com/activities/citable-code/`

[11] https://zenodo.org

permanent link [52] that does not depend on the long-term existence of GitHub.

2.5 Dependencies

Running any part of Clawpack requires a Python interpreter and the common Python packages numpy [30], `f2py` [48], matplotlib [27], as well as (except for the pure-Python 1D code) GNU make and a Fortran compiler. Other dependencies are optional, depending on which parts of Clawpack are to be used:

- IPython/Jupyter if using the notebook interfaces [47].
- PETSc [4], if using distributed parallelism in PyClaw.
- OpenMP, if using shared-memory parallelism in AMRClaw or GeoClaw.
- MATLAB, if using the legacy visualization tools.

3 Advances

This section describes the major changes in each of the code repositories in moving from Clawpack 4.x to the most recent version 5.3. A number of the repositories have seen only minor changes as the bulk of the development is focused on current research interests. There are a number of minor changes not listed here and the interested reader is encouraged to refer to the change logs[12] and the individual Clawpack Git repositories for a more complete list.

3.1 Global Changes

Substantial redesign of the Clawpack code base was performed in the move from Clawpack 4.x to 5.x. Major changes that affected all aspects of the code include:

- The interface to the Clawpack Riemann solvers was changed so that one set of solvers can be used for all versions of the code (including PyClaw via `f2py`[13]). Rather than appearing in scattered example directories, these Riemann solvers have all been collected into the new `riemann` repository. Modifications to the calling sequences were made to accommodate this increased generality.
- Calling sequences for a number of other Fortran subroutines were also modified based on experiences with the Clawpack 4.x code. These can also be used as a stand-alone product for those who only want the Riemann solvers.
- Python front-ends were redesigned to more easily specify run-time options for the solver and visualization. The Fortran variants (ClassicClaw, AMRClaw, and GeoClaw) all use a Python script to facilitate setting input variables. These scripts create text files with a rigidly specified format that are then read in when the Fortran code is run. The interface now allows updates to the input parameters while maintaining backwards compatibility.
- The indices of the primary conserved quantities were reordered. In Clawpack 4.x, the mth component of a system of equations in grid cell (i, j) (in two dimensions, for example), was stored in `q(i,j,m)`. In order to improve cache usage and to more easily interface with PETSc [4], a global change was made to the ordering so that the component number comes first; i.e. `q(m,i,j)`. A seemingly minor change like this affects a huge number of lines in the code and cannot easily be automated. The use of version control and regression tests was crucial in the successful completion of the project.

[12] http://www.clawpack.org/changes.html
[13] http://docs.scipy.org/doc/numpy-dev/f2py

3.2 Riemann: A Community-Driven Collection of Approximate Riemann Solvers

The methods implemented in Clawpack, and all modern Godunov-type methods for hyperbolic PDEs, are based on the solution of Riemann problems as discussed in Section 1.2. Whereas most existing codes for hyperbolic PDEs use Riemann solvers to compute fluxes, Clawpack Riemann solvers instead compute the waves (or discontinuities) that make up the Riemann solution. In the unsplit algorithm, Clawpack also makes use of *transverse* Riemann solvers, responsible for computing transport between cells that are only corner (in 2d) or edge (in 3d) adjacent.

For nonlinear systems, the exact solution of the Riemann problem is computationally costly and may involve both discontinuities (shocks and contact waves) and rarefactions. It is almost always preferable to employ inexact Riemann solvers that approximate the solution using discontinuities only, with an appropriate entropy condition. The solvers available in Clawpack are all approximate solvers, although one could easily implement their own exact solver and make it available in the format needed by Clawpack routines.

A common feature in all packages in the Clawpack suite is the use of a standard interface for Fortran Riemann solver routines. This ensures that new solvers or solver improvements developed for one package can immediately be used by all packages. To further facilitate this sharing and to avoid duplication, Riemann solvers are (with rare exceptions) not maintained under the other packages but are collected in a single repository named `riemann`. Users who develop new solvers for Clawpack are encouraged to submit them to the Riemann repository.

In the Fortran-based packages (Classic, AMRClaw, and GeoClaw) the Riemann solver is selected at compile-time by modifying a problem-specific Makefile. In PyClaw, the Riemann solver to be used is selected at run-time. This is made possible by compiling all of the Riemann solvers (when PyClaw is installed) and generating Python wrappers with `f2py`. For PyClaw, `riemann` also provides metadata (such as the number of equations, the number of waves, and the names of the conserved quantities) for each solver so that setup is made more transparent.

3.3 ClassicClaw

The `classic` repository contains code implementing the wave propagation algorithm on a single uniform grid, in much the same form as the original Clawpack 1.0 version of 1994 but with various enhancements added through the years. Following the introduction of Clawpack 4.4 the three-dimensional routines were left out of the Python user interfaces and plotting routines. These have been reintroduced in Clawpack 5. Additionally the OpenMP shared-memory parallelism capabilities have been extended to the three-dimensional code.

3.4 AMRClaw

Fortran code in the AMRClaw repository performs block-structured adaptive mesh refinement [6, 7] for both Clawpack and GeoClaw applications. The algorithms implemented in AMRClaw are discussed in detail in [9, 41], but a short description is given here to set the stage for a description of recent changes. This type of refinement solves the PDE on a hierarchy of logically rectangular grids. One (or more) level 1 grids comprise the entire domain, while grids at finer level are created and destroyed (as opposed to moving these grids) to follow important features in the solution.

AMRClaw includes the functionality for:

- Coordinating the flagging of points where refinement is needed, with a variety of criteria possible for flagging cells that need refinement from each level to the next finer level (including Richardson extrapolation, gradient testing, or user-specified criteria)[14],

[14]See http://www.clawpack.org/flag.html

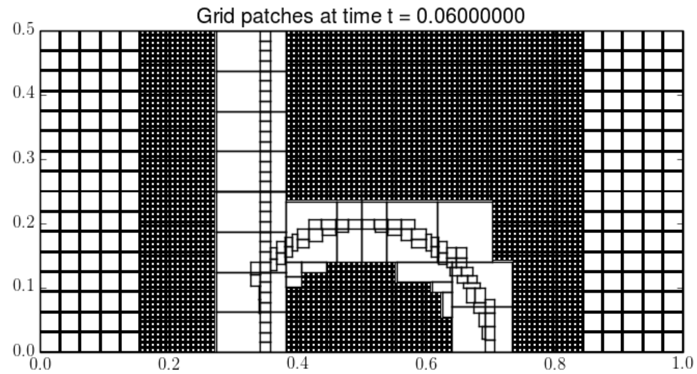


Figure 1: An illustration showing grid cells on levels one and two, and only grid outlines on levels three and four.

- Organizing the flagged points into efficient grid patches at the next finer level, using the algorithm of [11],
- Interpolating the solution to newly created fine grids and initializing auxiliary data (topography, wind velocity, metric data and so on) on these grids,
- Averaging fine grid solutions to coarser grids,
- Orchestrating the adaptive time stepping (i.e. sub-cycling in time),
- Interpolating coarse grid solution to fine grid ghost cells, and
- Maintaining conservation at patch boundaries between resolution levels.

AMRClaw now allows users to specify "regions" in space-time $[x_1, x_2] \times [y_1, y_2] \times [t_1, t_2]$ in which refinement is forced to be at least at some level L_1 and is allowed to be at most L_2. This can be useful for constraining refinement, e.g. allowing or ensuring resolution of only a small coastal region in a global tsunami simulation. Previously the user could enforce such conditions by writing a custom flagging routine, but now this is handled in a general manner so that the parameters above can all be specified in the Python problem specification. Multiple regions can be specified, and a simple rule is used to determine the constraints at a grid cell that lies in multiple regions.

Auxiliary arrays are often used in Clawpack to store data that describes the problem and the routine. The routine `setaux` must then be provided by the user to set these values each time a new grid patch is created. For some applications computing these values can be time-consuming. In Clawpack 5.2, this code was improved to allow reuse of values from previous patches at the same level where possible at each regridding time. This is backward compatible, since no harm is done if previously written routines are used that still compute and overwrite instead of checking a mask.

In Clawpack 5.3 the capability to specify spatially varying boundary conditions was added. For a single grid, it is a simple matter to compute the location of the ghost cells that extend outside the computational domain and set them appropriately. With AMR however, the boundary condition routine can be called for a grid located anywhere in the domain, and may contain fewer or larger numbers of ghost cells. For this reason, the boundary condition routines do not assume a fixed number of ghost cells.

Anisotropic refinement is allowed in both two and three dimensions. This means that the spatial and temporal refinement ratios can be specified independently from one another (as long as the temporal refinement satisfies the CFL condition). In addition, capabilities have been added to automatically select the refinement ratio in time on each level based on the CFL condition. This has only been

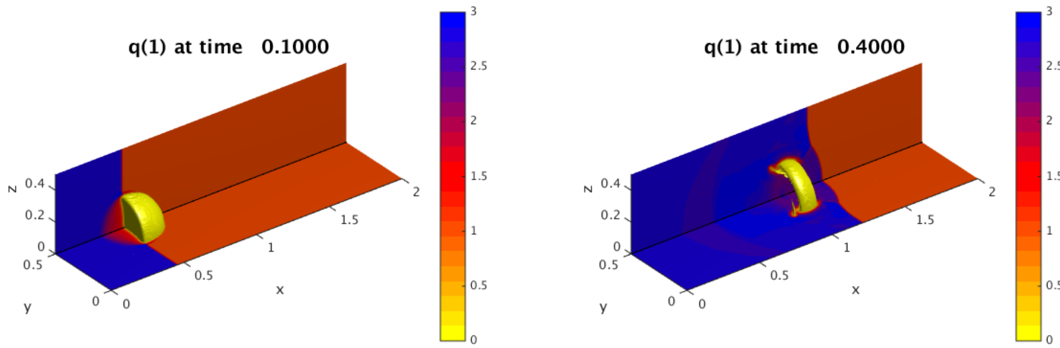


Figure 2: AMRCLAW example demonstrating a shock-bubble interaction in the Euler equations of compressible gas-dynamics at two times, illustrating the need for adaptive refinement to capture localized behavior. There are two $20 \times 10 \times 10$ grids at level 1. They are refined where needed by factors of 4 and then 2 in this 3-level run.

implemented in GEOCLAW where the wave speed in the shallow water equations depends on the local depth. The finest grids are often located only in shallow coastal regions, so a large refinement ratio in space does not lead to a large refinement ratio in time.

AMRCLAW has been parallelized using OpenMP directives. The main paradigm in structured AMR is an outer loop over levels of refinement, and in inner loop overall grids at that level, where the same operation is performed on each grid (i.e. taking a time step, finding ghost cells, conservation updates, etc.). This inner loop is parallelized using a `parallel for` loop construct one thread is assigned to operate on one grid. Dynamic scheduling is used with a chunk size of one. To help with load balancing, grids at each level are sorted from largest to smallest, using the total number of cells in the grid as an indicator of work. In addition there is a grids are limited to a maximum of 32 cells in each dimension, otherwise they are bisected until this condition is met. Note that this approach causes a memory bulge. Each thread must have its own scratch arrays to save the incoming and outgoing waves and fluxes for future conservation fix-ups. The bulge is directly proportional to the number of threads executing. For stack-based memory allocation per thread, the use of the environment variable `OMP_STACKSIZE` to increase the limit may be necessary.

Fig. 2 shows two snapshots of the solution to a three-dimensional shock-bubble interaction problem found in the CLAWPACK `apps` repository, illustrating localized phenomena requiring adaptive refinement. In Fig. 3 we show scalability tests and some timings for this example, when run on a 40 core Intel Xeon Haswell machine (E5-2670v3 at 2.3 GHz), using `KMP_AFFINITY compact` with one thread per core. For timing purposes, the only modifications made to the input parameters was to turn off check-pointing and graphics output. The plot on the left shows that most of the wall clock time is in the integration routine (`stepgrid`), which closely tracks the total time. The second chunk of time is in the regridding, which contains algorithms that are not completely scalable. Very little time is in the filling of ghost cells, mostly from other patches but also includes those at domain boundaries. The efficiency is above 80% until 24 cores, then drops off dramatically. Note that there are only two grids on level, and an average of 22.8 level 2 grids. Most of the work is on level 3 grids, where there are an average of 138.1 grids over all the level 3 timestep. This is very coarse for large numbers of cores (hence the dropoff in efficiency). At 40 cores, there are less than 4 grids per core, and the grids are very different sizes.

The target architecture for AMRCLAW and GEOCLAW are multi-core machines. PYCLAW on the other hand scales to tens of thousands of cores using MPI via PETSc [4] but is not adaptive.

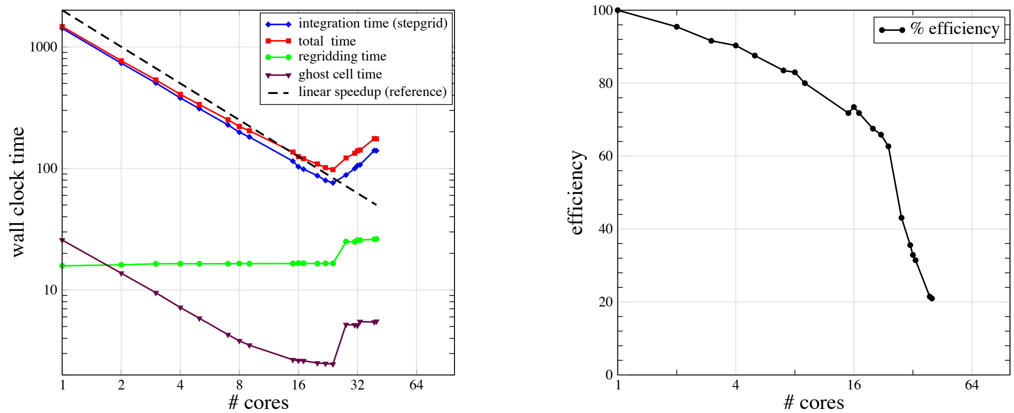


Figure 3: Left is strong scaling results for the AMRClaw example shown in Fig. 2. Right is plot of efficiency based on total computational time.

3.5 GeoClaw

The GeoClaw branch of Clawpack was developed to solve the two-dimensional shallow water equations over topography for modeling tsunami generation, propagation, and inundation. The AMRClaw code formed the starting point but it was necessary to make many modifications to support the requirements of this application, as described briefly below. This code originated with the work of George [19, 20, 21] and was initially called TsunamiClaw. Later it became clear that many other geophysical flow applications have similar requirements and the code was generalized as GeoClaw.

One of the major issues is the treatment of wetting and drying of grid cells at the margins of the flow. The handling of dry states in a Riemann solver is difficult to handle robustly, and has gone through several iterations. GeoClaw must also be well-balanced in order to preserve steady states, in particular the “ocean at rest”. To achieve this, the source terms in the momentum equations arising from variations in topography are incorporated into the Riemann solver rather than using a fractional step splitting approach. This is critical for modeling waves that have very small amplitudes relative to the variations in the depth of the ocean. See [40] for a general discussion of such methods and [20, 21] for details of the Riemann solver used in GeoClaw. Other features of GeoClaw include the ability to solve the equations in latitude–longitude coordinates on the surface of the sphere, and the incorporation of source terms modeling bottom friction using a Manning formulation. More details about the code and tsunami modeling applications can be found in [8, 41]. In 2011, a significant effort took place to verify and validate GeoClaw against the US National Tsunami Hazard Mitigation Program (NTHMP) benchmarks [25]. NTHMP approval of the code allows GeoClaw to be used in hazard mapping projects that are funded by this program or other federal and state agencies, e.g. [23, 24]. One such project is illustrated in Fig. 4.

In addition to a variety of tsunami modeling applications, GeoClaw has been used to solve dam break problems in steep terrain [18], storm surge problems [44] (see Fig. 5), and submarine landslides [34]. The code also formed the basis for solving the multi-layer shallow water equations for storm surge modeling [42, 43], and is currently being extended further to handle debris flow modeling in the packages D-Claw [28, 22] (see Figs. 6 and 7).

Nearly one quarter of the files in the AMRClaw source library have to be modified for GeoClaw. There are currently 113 files in the AMRClaw 2D library, of which 26 are replaced by a GeoClaw-specific files of the same name in the GeoClaw 2D library. For example, to preserve a flat sea surface when interpolating, it is necessary to interpolate the surface elevation (topography plus water depth) rather than simply interpolating the depth component of the solution vector as would normally be done in AMRClaw. An additional 24 files in the GeoClaw shallow water equations library handle other complications introduced by the need to model tsunamis and storm surge.

Several other substantial improvements in the algorithms implemented in GeoClaw have been made between versions 4.6 and 5.3.0, including:

- In depth-averaged flow, the wave speed and therefore the CFL condition depends on the depth. As a result, flows in shallow water that have been refined spatially may not need to be refined in time. This "variable-time-stepping" was easily added along with the anisotropic capabilities that were added to AMRClaw.

- The ability to specify topography via a set of `topo` files that may cover overlapping regions at different resolutions has been added. The finite volume method requires cell averages of topography, computed by integrating a piecewise bilinear function constructed from the input `topo` files over each grid cell. In Clawpack 5.1.0, this was improved to allow an arbitrary number of nested `topo` grids. When adaptive mesh refinement is used, regridding may take place every few time steps. Improvements were made in 5.2.0 so that topography could be copied rather than always being recomputed in regions where there is an existing old grid.

- The user can now provide multiple `dtopo` files that specify changes to the initial topography at a series of times. This is used to specify sea-floor motion during a tsunamigenic earthquake, but can also be used to specify submarine landslide motion or a failing dam, for example.

- A number of new Python modules has been developed to assist the user in working with `topo` and `dtopo` files. These are documented in the Clawpack documentation and several of them are illustrated with Jupyter notebooks found in the Clawpack Gallery.

- New capabilities were added in 5.0.0 to monitor the maximum of various flow quantities over a specified time range of a simulation. This capability is crucial for many applications where the maximum flow depth at each point, maximum current velocities in a harbor, or maximum momentum flux (a measure of the hydrodynamic force that would be exerted by the flow on a structure) is desired. Arrival time of the first wave at each point can also be monitored. Such capabilities were included in the 4.x version of the code, but were more limited and did not always perform properly near the edges of refinement patches. In Version 5.2 these routines were further improved and extended. The user can specify a grid of points on which to monitor values, and the new code is more flexible in allowing one-dimensional grids (e.g. a transect), two-dimensional rectangular grids, or an arbitrary set of points[15].

3.6 PyClaw

PyClaw is an object-oriented Python package that provides a convenient way to set up problems and call the algorithms of Clawpack. It grew from what was initially a set of data structures and file IO routines that are used by the other Clawpack codes and by VisClaw. These routines were released in an early form in later 4.x versions of Clawpack. Those releases also included a fully-functional implementation of the 1D classic algorithm in pure Python. That implementation still exists in PyClaw and is useful for understanding the algorithm.

The current release of PyClaw includes access to the classic algorithms as well as the high-order algorithms introduced in SharpClaw [32] (i.e., WENO reconstruction and Runge–Kutta integrators) and can be used on large distributed-memory parallel machines. For the latter capability, PyClaw relies on PETSc [4]. Lower-level code (whatever gets executed repeatedly and needs to be fast) from the earlier Fortran Classic and SharpClaw codes is automatically wrapped at install time using `f2py`.

Recent applications of PyClaw include studies of laser light trapping by moving refractive index perturbations [51], instabilities of weakly nonlinear detonation waves [17], and effective dispersion of nonlinear waves via diffraction in periodic materials [33]. Two of these are depicted in Fig. 8.

[15] Described in http://www.clawpack.org/fgmax.html

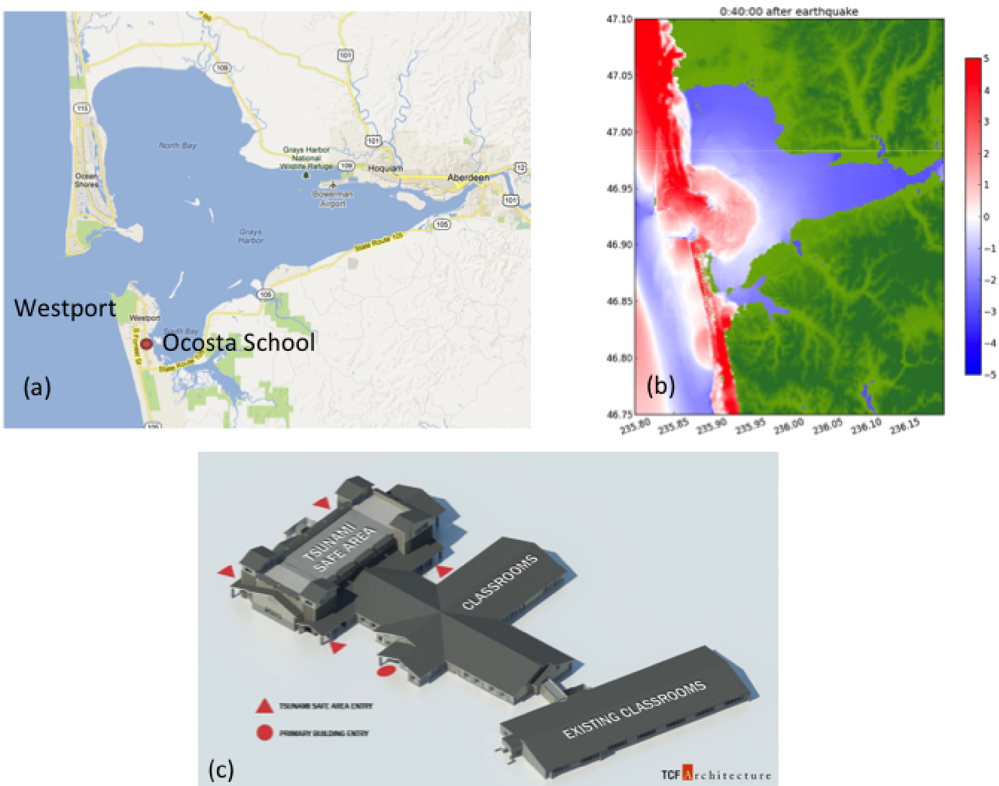
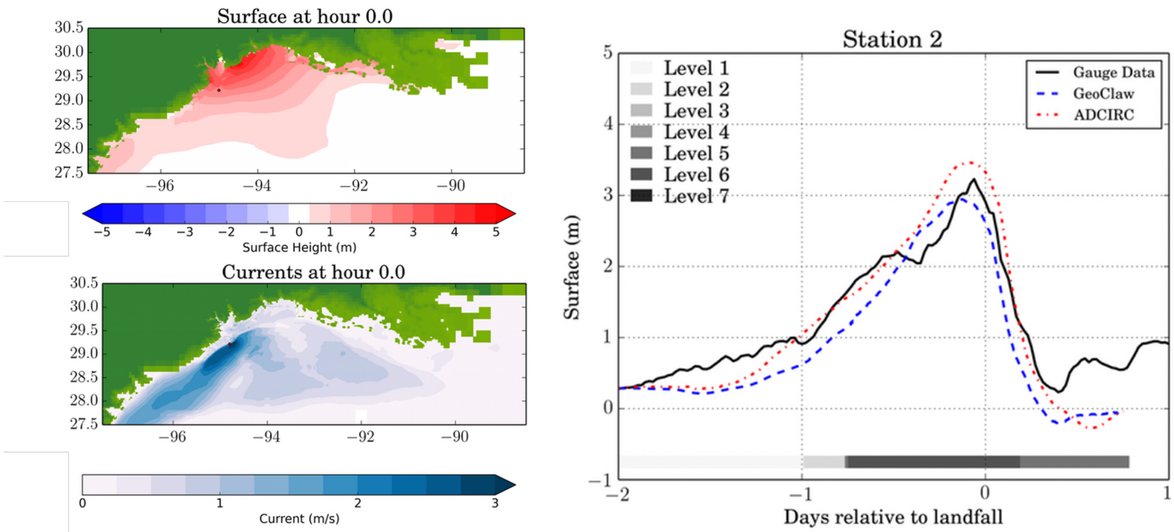


Figure 4: Gray's Harbor showing Westport, WA on southern peninsula. (Google map data and image, 2016.) (b) Simulation of a potential magnitude 9 Cascadia Subduction Zone event, 40 minutes after the earthquake. (c) Design for new Ocosta Elementary School in Westport, based in part on GeoClaw simulations [23]. Image courtesy of TCF Architecture.



Package	Threads	Wall Time	Core Time
ADCIRC	4000	35 minutes	2333 hours
GeoClaw	4	2 hours	8 hours

Figure 5: Top Left: A snapshot of a GeoClaw storm surge simulation of Hurricane Ike at landfall. Top Right: Tide gauge data computed from GeoClaw and ADCIRC along with observed data at the same location. Bottom: Computational effort and timings for GeoClaw and ADCIRC. From [44].

3.6.1 Librarization and extensibility

Scientific software is easier to use, extend, and integrate with other tools when it is designed as a library [12]. Clawpack has always been designed to be extensible, but PyClaw takes this further in several ways. First, it is distributed via a widely-used package management system, `pip`. Second, the default installation process (“`pip install clawpack`”) provides the user with a fully-compiled code and does not require setting environment variables. Like other Clawpack packages, PyClaw provides several “hooks” for users to plug in custom routines (for instance, to specify boundary conditions). In PyClaw, these routines – including the Riemann solver itself – are selected at run-time, rather than at compile-time. These routines can be written directly in Python, or (if they are performance-critical) in a compiled language (like Fortran or C) and wrapped with one of the many available tools. Problem setup (including things like initial conditions, algorithm selection, and output specification) is also performed at run-time, which means that researchers can bypass much of the slower code-compile-execute-post-process cycle. It is intended that PyClaw be easily usable within other packages (without control of `main()`).

3.6.2 Python geometry

PyClaw includes Python classes for describing collections of structured grids and data on them. These classes are also used by the other codes and VisClaw, for post-processing. A mesh in Clawpack always consists of a set of (possibly mapped) tensor-product grids (interval, quadrilateral, or hexahedral), also referred to as patches. At present, PyClaw solvers operate only on a single patch, but the geometry and grids already incorporate multi-patch capabilities for visualization in AMRClaw and GeoClaw.

3.6.3 PyClaw solvers

PyClaw includes an interface to both the Classic solvers (already described above) and those of SharpClaw [31]. SharpClaw uses a traditional method-of-lines approach to achieve high-order resolution in

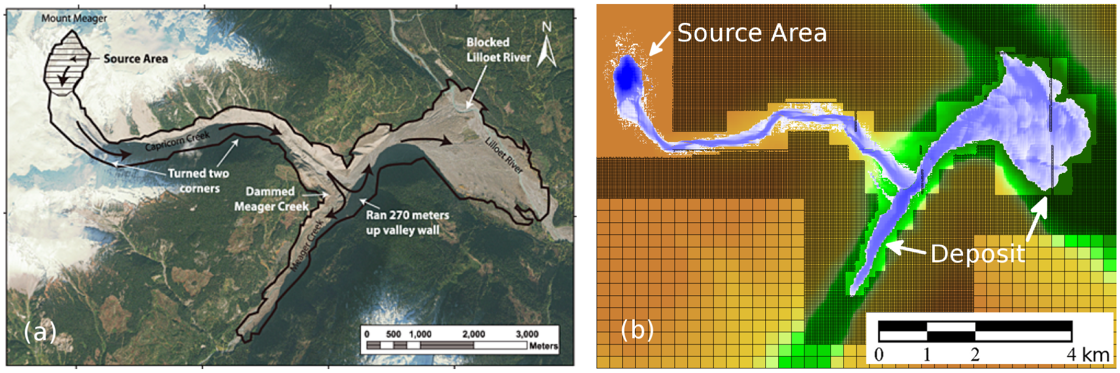


Figure 6: (a) Photograph of the 2010 Mt. Meager debris-flow deposit, from [2]. (b) Simulated debris flow, from D. George.

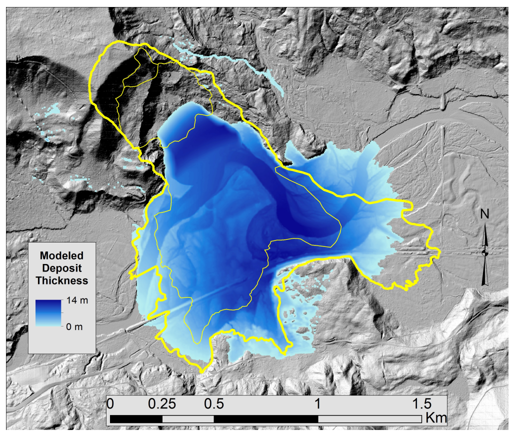


Figure 7: Observed (yellow line) and computed (blue) landslide at Oso, WA in 2014 [29].

space and time. Spatial operators are discretized first, resulting in a system of ODEs that is then solved using Runge–Kutta or linear multistep methods. The spatial derivatives are computed using a weighted essentially non-oscillatory (WENO) reconstruction from cell averages, which suppresses spurious oscillations near discontinuities. The WENO routines in SharpClaw were generated by PyWENO[16], which is a standalone package that generates WENO routines.

The default time stepping routines in SharpClaw are strong stability preserving (SSP) Runge–Kutta methods of order two to four. Some of the methods use extra stages in order to allow more efficient time stepping with larger CFL numbers. Time stepping in SharpClaw has recently been augmented to include linear multistep methods with variable step size. These methods use a time step size selection that ensures the strong stability preserving property, as described in [26].

3.6.4 Parallelism

PyClaw includes a distributed parallel backend that uses PETSc through the Python wrapper `petsc4py`. The parallel code uses the same low-level routines without modification. In the high-level routines, only a few hundred lines of Python code deal explicitly with parallel communication, in order to transfer ghost cell information between subdomains and to find the global maximum CFL number in order to adapt the time step size. For instance, the computation shown in the right part of Fig. 8 involved more

[16] http://github.com/memmett/PyWENO

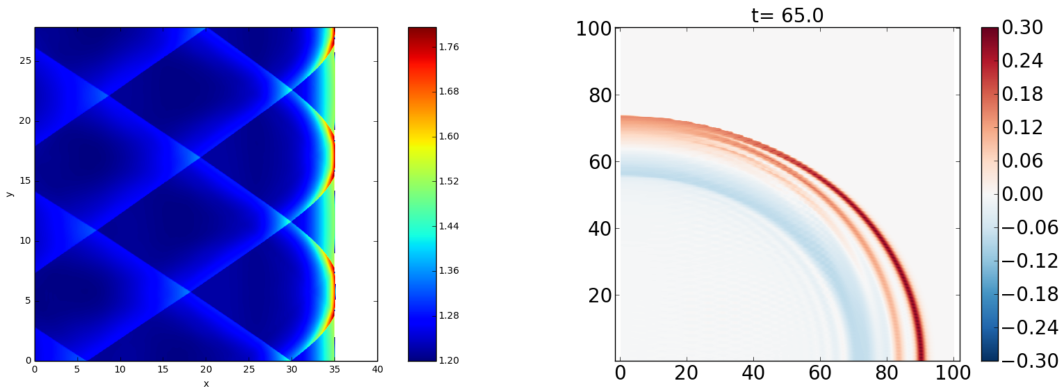


Figure 8: Left: A two-dimensional detonation wave solution of the reactive Euler equations, showing transverse shocks that arise from instabilities; see [17]. Right: Dispersion of waves in a layered medium with matched impedance and periodically-varying sound speed; see [33].

than 120 million degrees of freedom and was run on two racks of the Shaheen I BlueGene/P supercomputer. The code has been demonstrated to scale with better than 90% efficiency in even larger tests on tens of thousands of processors on both the Shaheen I (BlueGene/P) and Shaheen II (Cray XC40) supercomputers at KAUST. A hybrid MPI/OpenMP version is already available in a development branch and will be included in future releases.

3.7 VisClaw : Visualizing Clawpack output

A practical way to visualize the results of simulations is essential to any software package for solving PDEs. This is particularly true for simulations making use of adaptive mesh refinement, since most available visualization packages do not have tools that conveniently visualize hierarchical AMR data. VisClaw provides support for all of the main Clawpack submodules, including ClassicClaw, AMRClaw, PyClaw and GeoClaw.

From the first release in 1994, Clawpack has included tools for visualizing the output of Clawpack and AMRClaw runs. Up until the release of version Clawpack 4.x, these visualization tools consisted primarily of Matlab routines for creating one, two and three dimensional plots including pseudo-color plots, Schlieren plots, contour plots and scatter plots, including radially or spherically symmetric data. Built-in tools were also available for handling one, two and three-dimensional mapped grids. Starting with version 4.x, however, it was recognized that a reliance on proprietary software for visualization prevented a sizable potential user base from making use of the Clawpack software. The one and two dimensional plotting routines were converted from Matlab to matplotlib, a popular open source Python package for producing publication quality graphics for one and two dimensional data [27].

With the development of Clawpack Version 5 and above, Python graphics tools have been collected into the VisClaw repository. The VisClaw tools extend the functionality of the Version 4.x Python routines for creating one and two dimensional plots, and adds several new capabilities. Chief among these are the ability to generate output to webpages, where a series of plots can be viewed individually or as an animated time sequence using the Javascript package[17] (which was motivated by code in an earlier version of Clawpack). The VisClaw module `Iplotclaw` provides interactive plotting capabilities from the Python or IPython prompt. Providing much of the same interactive capabilities as the original Matlab routines, `Iplotclaw` allows the user to step, interactively, through a time sequence of plots, jump from one frame to another, or interactively explore data from the current time frame.

[17] https://github.com/jakevdp/JSAnimation

3.7.1 Tools for visualizing geo-spatial data produced by GeoClaw

The geo-spatial data generated by GeoClaw has particular visualization requirements. Tsunami or storm surge simulations are most useful when the plots showing inundation or flooding levels are overlaid onto background bathymetry or topography. Supplementary one dimensional time series data (e.g. gauge data) numerically interpolated from the simulation at fixed spatial locations are most useful when compared graphically to observational data. Finally, to more thoroughly analyze the computational data, simulation data should be made available in formats that can be easily exported to GIS tools such as ArcGIS[18] or the open source alternative QGIS[19]. For exploration of preliminary results or communicating results to non-experts, Google Earth is also helpful.

The latest release of Clawpack includes many specialized VisClaw routines for handling the above issues with plotting geo-spatial data. Topography or bathymetry data that was used in the simulation will be read by the graphing routines, and, using distinct colormaps, both water and land can be viewed on the same plot. Additionally, gauge locations can be added, along with contours of water and land. One dimensional gauge plots are also created, according to user-customizable routines. In these gauge plotting routines, users can easily include observational data to compare with GeoClaw simulation results.

In addition to HTML and Latex formats available for all Clawpack results, VisClaw will now also produce KML and KMZ files suitable for visualizing results in Google Earth. Using the same `matplotlib` graphics routines, VisClaw creates PNG files that can be used as GroundOverlay features in a KML file. Other features, such as gauges, borders on AMR grids, and user specified regions can also be shown on Google Earth. All KML and PNG files are compressed into a single KMZ file that can be opened directly in Google Earth or made available on-line. While VisClaw does not have any direct support for ArcGIS or QGIS, the KML files created for Google Earth can be edited for export, along with associated PNG files to these other GIS applications.

3.7.2 Matlab plotting routines

The Matlab plotting tools available in early versions of Clawpack are still included in VisClaw. While most of the one and two dimensional capabilities available originally in the Matlab suite have been ported to Python and `matplotlib`, the original Matlab routines are still available in the Matlab suite of plotting tools. Other plotting capabilities, such as two dimensional manifolds embedded in three dimensional space, or three dimensional plots of fully three-dimensional data are only available in the Matlab routines in a way that interfaces directly with Clawpack. More advanced three-dimensional plotting capabilities are planned for future releases of VisClaw.

4 Conclusions

Clawpack has evolved over the past 20 years from its genesis as a small and focused software package that two core developers could manage without version control. It is now an ecosystem of related projects that share a core philosophy and some common code (notably Riemann solvers and visualization tools), but that are aimed at different user communities and that are developed by overlapping but somewhat distinct groups of developers scattered at many institutions. The adoption of better software engineering practices, in particular the use of Git and GitHub as an open development platform and the use of pull requests to discuss proposed changes, has been instrumental in facilitating the development of many of the new capabilities summarized in this paper. These developer facing improvements of course affect the user as well since better and faster development cycles means better and faster implementation of features. The user facing features already implemented in version 5 have opened up the use of Clawpack to a broader audience.

[18] http://www.arcgis.com

[19] http://www.qgis.org

4.1 Future Plans

The Clawpack development team continues to look forward to new ideas and efforts that will allow great accessibility to the project as well as new capabilities that the core development team has not thought of. To this end a number of the broad efforts that are being considered for the next major release of Clawpack include

- An increased librarization effort with the Fortran based sub-packages,
- An extensible and more accessible interface to the Riemann solvers,
- An effort to allow PyClaw and the Clawpack Fortran packages to rely on more of the same code-base,
- An increased emphasis on a larger development community,
- More support for new frameworks such as ForestClaw [13],
- Adjoint error estimation for flagging cells to increase the efficiency of the AMR codes [15],
- A refactoring of the visualization tools in VisClaw, along with support for additional backends, particularly for three-dimensional results (e.g. `Mayavi`[20], `VisIt`[21], `ParaView`[22], or `yt`[23]).

Acknowledgements

We wish to thank the many people who have made contributions to the Clawpack software over the years, including users who have submitted bug reports (or even better, bug fixes!) or have suggested improvements to the software.

References

[1] M. Adams, P. Colella, J. N. Johnson, N. D. Keen, T. J. Ligocki, D. F. Martin, P. W. McCorquodale, D. Modiano, P. O. Schwartz, T. D. Sternberg, and B. Van Straalen, Chombo Software Package for AMR Applications - Design Document, Tech. Report LBNL-6616E, Lawrence Berkeley National Laboratory.

[2] K. Allstadt, *Extracting source characteristics and dynamics of the August 2010 Mount Meager landslide from broadband seismograms*, J. Geophys. Res., 118 (2013), pp. 1472–1490, doi:10.1002/jgrf.20110.

[3] R. W. Anderson, W. J. Arrighi, N. S. Elliott, B. T. N. Gunney, and R. Hornung, *SAMRAI*, tech. report.

[4] S. Balay, J. Brown, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, *PETSc Users Manual*, Tech. Report ANL-95/11 - Revision 3.1, Argonne National Laboratory, 2010, http://www.mcs.anl.gov/petsc.

[5] D. Bale, R. J. LeVeque, S. Mitran, and J. A. Rossmanith, *A wave-propagation method for conservation laws and balance laws with spatially varying flux functions*, SIAM J. Sci. Comput., 24 (2002), pp. 955–978, http://www.amath.washington.edu/~rjl/pubs/vcflux/.

[20] http://docs.enthought.com/mayavi/mayavi/
[21] https://visit.llnl.gov
[22] http://www.paraview.org/
[23] http://yt-project.org/

[6] M. Berger and J. Oliger, *Adaptive mesh refinement for hyperbolic partial differential equations*, J. Comput. Phys., 53 (1984), pp. 484–512.

[7] M. J. Berger and P. Colella, *Local adaptive mesh refinement for shock hydrodynamics*, J. Comput. Phys., 82 (1989), pp. 64–84.

[8] M. J. Berger, D. L. George, R. J. LeVeque, and K. T. Mandli, *The GeoClaw software for depth-averaged flows with adaptive refinement*, Adv. Water Res., 34 (2011), pp. 1195–1206, www.clawpack.org/links/papers/awr11.

[9] M. J. Berger and R. J. LeVeque, *Adaptive mesh refinement using wave-propagation algorithms for hyperbolic systems*, SIAM J. Numer. Anal., 35 (1998), pp. 2298–2316, http://epubs.siam.org/sam-bin/dbq/article/31597.

[10] M. J. Berger and R. J. LeVeque, *Adaptive Mesh Refinement Using Wave-Propagation Algorithms for Hyperbolic Systems*, SIAM Journal on Numerical Analysis, 35 (1998), pp. 2298–2316, doi:10.2307/2587259.

[11] M. J. Berger and I. Rigoutsos, *An algorithm for point clustering and grid generation*, IEEE Trans. Sys. Man & Cyber., 21 (1991), pp. 1278–1286.

[12] J. Brown, M. G. Knepley, and B. F. Smith, *Run-Time Extensibility and Librarization of Simulation Software*, Computing in Science & Engineering, 17 (2015), pp. 38–45, doi:10.1109/MCSE.2014.95, http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6894481.

[13] C. Burstedde, D. A. Calhoun, K. Mandli, and A. R. Terrel, *ForestClaw: Hybrid forest-of-octrees AMR for hyperbolic conservation laws*, in Parallel Computing : Accelerating Computational Science and Engineering (CSE), M. Bader, A. Bode, H.-J. Bungartz, M. Gerndt, G. R. Joubert, and F. Peters, eds., vol. 25 of Advances in Parallel Computing, IOS Press, 2014, pp. 253–262.

[14] Clawpack Development Team, *Clawpack software*, 2015, http://www.clawpack.org. Version 5.3.0.

[15] B. N. Davis and R. J. LeVeque, *Adjoint methods for guiding adaptive mesh refinement in wave propagation problems*. http://arxiv.org/abs/1511.03645, 2015.

[16] R. Deiterding, *Block-structured Adaptive Mesh Refinement - Theory, Implementation and Application*, ESAIM: Proceedings, 34 (2011), pp. 97–150, doi:10.1051/proc/201134002, http://www.esaim-proc.org/10.1051/proc/201134002.

[17] L. M. Faria and A. R. Kasimov, *Qualitative modeling of the dynamics of detonations with losses*, Proceedings of the Combustion Institute, 35 (2015).

[18] D. George, *Adaptive finite volume methods with well-balanced Riemann solvers for modeling floods in rugged terrain: Application to the Malpasset dam-break flood (France, 1959)*, Int. J. Numer. Meth. Fluids, (2010), doi:10.1002/fld.2298, http://www3.interscience.wiley.com/journal/123309491/abstract.

[19] D. L. George, *Numerical approximation of the nonlinear shallow water equations with topography and dry-beds: A Godunov-Type scheme*, master's thesis, University of Washington, 2004.

[20] D. L. George, *Finite Volume Methods and Adaptive Refinement for Tsunami Propagation and Inundation*, PhD thesis, University of Washington, 2006.

[21] D. L. George, *Augmented Riemann solvers for the shallow water equations over variable topography with steady states and inundation*, J. Comput. Phys., 227 (2008), pp. 3089–3113.

[22] D. L. George and R. M. Iverson, *A depth-averaged debris-flow model that includes the effects of evolving dilatancy. II. Numerical predictions and experimental tests*, Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences, 470 (2014), pp. 20130820–20130820, doi:10.1098/rspa.2013.0820, http://rspa.royalsocietypublishing.org/cgi/doi/10.1098/rspa.2013.0820.

[23] F. I. González, R. J. LeVeque, and L. M. Adams, *Tsunami Hazard Assessment of the Ocosta School Site in Westport, WA*, tech. report, Washington State Emergency Management Division, Sept. 2013, https://digital.lib.washington.edu/researchworks/handle/1773/24054 (accessed 2015-04-28).

[24] F. I. Gonzalez, R. J. LeVeque, L. M. Adams, C. Goldfinger, G. R. Priest, and K. Wang, *Probabilistic Tsunami Hazard Assessment (PTHA) for Crescent City, CA*, Sept. 2014, https://digital.lib.washington.edu/researchworks/handle/1773/25916 (accessed 2015-10-11).

[25] F. I. González, R. J. LeVeque, J. Varkovitzky, P. Chamberlain, B. Hirai, and D. L. George, *GeoClaw results for the NTHMP tsunami benchmark problems*, 2011. http://depts.washington.edu/clawpack/links/nthmp-benchmarks/.

[26] Y. Hadjimichael, D. I. Ketcheson, L. Lóczi, and A. Németh, *Strong stability preserving explicit linear multistep methods with variable step size.* Submitted. Preprint available at http://arxiv.org/abs/1504.04107., http://arxiv.org/abs/1504.04107.

[27] J. D. Hunter, *Matplotlib: A 2d graphics environment*, Computing In Science & Engineering, 9 (2007), pp. 90–95.

[28] R. M. Iverson and D. L. George, *A depth-averaged debris-flow model that includes the effects of evolving dilatancy. I. Physical basis*, Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences, 470 (2014), pp. 20130819–20130819, doi:10.1098/rspa.2013.0819.

[29] R. M. Iverson, D. L. George, K. Allstadt, M. E. Reid, B. D. Collins, J. W. Vallance, S. P. Schilling, J. W. Godt, C. M. Cannon, C. S. Magirl, R. L. Baum, J. A. Coe, W. H. Schulz, and J. B. Bower, *Landslide mobility and hazards: implications of the 2014 Oso disaster*, Earth and Planetary Science Letters, 412 (2015), pp. 197–208, doi:10.1016/j.epsl.2014.12.020.

[30] E. Jones, T. Oliphant, P. Peterson, et al., *SciPy: Open source scientific tools for Python*, 2001–, http://www.scipy.org/. [Online; accessed 2016-05-13].

[31] D. I. Ketcheson, K. Mandli, A. J. Ahmadia, A. Alghamdi, M. Q. de Luna, M. Parsani, M. G. Knepley, and M. Emmett, *Pyclaw: Accessible, extensible, scalable tools for wave propagation problems*, SIAM Journal on Scientific Computing, 34 (2012), pp. C210–C231.

[32] D. I. Ketcheson, M. Parsani, and R. J. LeVeque, *High-order Wave Propagation Algorithms for Hyperbolic Systems*, SIAM Journal on Scientific Computing, 35 (2013), pp. A351–A377, doi:10.1137/110830320, arXiv:1111.3499.

[33] D. I. Ketcheson and M. Quezada de Luna, *Diffractons: 2D solitary waves in layered periodic media*, Multiscale Modeling & Simulation, 13 (2015), pp. 440–458.

[34] J. Kim, *Finite volume methods for Tsunamis genereated by submarine landslides*, PhD thesis, University of Washington, 2014. http://hdl.handle.net/1773/25374.

[35] J. O. Langseth, *Wave Propagation Schemes, Operator Splittings, and Front Tracking for Hyperbolic Conservation Laws*, PhD thesis, Department of Informatics, University of Oslo, 1996.

[36] J. O. Langseth and R. J. LeVeque, *A wave-propagation method for three-dimensional hyperbolic conservation laws*, J. Comput. Phys., 165 (2000), pp. 126–166, http://www.amath.washington.edu/~rjl/pubs/wp3d/.

[37] R. J. LeVeque, *Clawpack v1.0 announcement*. http://www.netlib.org/na-digest-html/94/v94n44.html#5, 1994.

[38] R. J. LeVeque, *Wave propagation algorithms for multi-dimensional hyperbolic systems*, J. Comput. Phys., 131 (1997), pp. 327–353, http://www.amath.washington.edu/~rjl/pubs/wpalg/.

[39] R. J. LeVeque, *Finite Volume Methods for Hyperbolic Problems*, Cambridge University Press, 2002, http://amath.washington.edu/~claw/book.html.

[40] R. J. LeVeque, *A well-balanced path-integral f-wave method for hyperbolic problems with source terms*, J. Sci. Comput., 48 (2010), pp. 209–226, doi:10.1007/s10915-010-9411-0.

[41] R. J. LeVeque, D. L. George, and M. J. Berger, *Tsunami modeling with adaptively refined finite volume methods*, Acta Numerica, (2011), pp. 211–289.

[42] K. T. Mandli, *Finite Volume Methods for the Multilayer Shallow Water Equations with Applications to Storm Surges*, PhD thesis, University of Washington, 2011.

[43] K. T. Mandli, *A Numerical Method for the Two Layer Shallow Water Equations with Dry States*, Ocean Modelling, 72 (2013), pp. 80–91, doi:10.1016/j.ocemod.2013.08.001.

[44] K. T. Mandli and C. N. Dawson, *Adaptive Mesh Refinement for Storm Surge*, Ocean Modelling, 75 (2014), pp. 36–50, http://www.sciencedirect.com/science/article/pii/S1463500314000031.

[45] MATLAB, *version 8.5.0 (R2015a)*, The MathWorks Inc., Natick, Massachusetts, 2015a.

[46] OpenFOAM Foundation, *OpenFOAM*, http://openfoam.org/.

[47] F. Pérez and B. E. Granger, *Ipython: A system for interactive scientific computing*, Computing in Science and Engineering, 9 (2007).

[48] P. Peterson, *F2PY: a tool for connecting Fortran and Python programs*, International Journal of Computational Science and Engineering, 4 (2009), http://dl.acm.org/citation.cfm?id=1647766.

[49] H. A. Piwowar, R. S. Day, and D. B. Fridsma, *Sharing detailed research data is associated with increased citation rate*, PloS One, 2 (2007), p. e308.

[50] S. Popinet, *The Gerris Flow Solver*, (2001).

[51] D. San Roman Alerigi, *Exploring Heterogeneous Time-Varying Materials for Photonic Applications, Towards Solutions for the Manipulation and Confinement of Light*, PhD thesis, King Abdullah University of Science and Technology, 2015.

[52] C. D. Team, *Clawpack version 5.3.1*, Nov. 2015, doi:10.5281/zenodo.50982, http://dx.doi.org/10.5281/zenodo.50982.

[53] M. J. Turk, *Scaling a code in the human dimension*, ACM, July 2013, doi:10.1145/2484762.2484782.

[54] P. Vandewalle, *Code sharing is associated with research impact in image processing*, Comput. Sci. Eng., 14 (2012), pp. 42–47.