PetClaw: A Scalable Parallel Nonlinear Wave Propagation Solver for Python

Amal Alghamdi, Aron Ahmadia, David I. Ketcheson
King Abdullah University of Science & Technology

Matthew G. Knepley
University of Chicago

Kyle T. Mandli
University of Washington

Lisandro Dalcin
CIMEC

Keywords: wave propagation, PETSc, petsc4py, Clawpack, PyClaw

Abstract

We present **PetClaw**, a scalable distributed-memory solver for time-dependent nonlinear wave propagation. **PetClaw** unifies two well-known scientific computing packages, Clawpack and PETSc, using Python interfaces into both. We rely on Clawpack to provide the infrastructure and kernels for time-dependent nonlinear wave propagation. Similarly, we rely on PETSc to manage distributed data arrays and the communication between them. We describe both the implementation and performance of **PetClaw** as well as our challenges and accomplishments in scaling a Python-based code to tens of thousands of cores on the BlueGene/P architecture. The capabilities of **PetClaw** are demonstrated through application to a novel problem involving elastic waves in a heterogeneous medium. Very finely resolved simulations are used to demonstrate the suppression of shock formation in this system.

1. INTRODUCTION

Numerical solvers for systems of hyperbolic conservation laws are an important tool in the computational scientist's toolbox. Clawpack is a widely used, state-of-the-art package for solving hyperbolic systems of equations. It has been used to solve problems in astrophysics, geodynamics, magnetohydrodynamics, oceanography, porous media flow, and numerous other application areas. In this work, we present a parallel extension of Clawpack based on incorporation of several existing tools. Efforts have been made previously to parallelize Clawpack for distributed memory architectures, demonstrating a general need for a scalable version of the software. The clear standard for developing a scalable Clawpack implementation is the Message Passing Interface, which is universally supported by all major supercomputers. Instead of writing a direct interface into the MPI libraries, we make use of the Portable Extensible Toolkit for Scientific Computing (PETSc) to provide a layer of abstraction between the discretized systems of equations and their mapping to distributed processes. We use the novel approach of unifying two large scientific libraries through their Python interfaces (**PyClaw** and **petsc4py**), explicitly relying on **numpy** to seamlessly migrate numerical data between Python, C, and Fortran.

The paper is organized as follows. In section 2., we describe the physical and numerical structure of hyperbolic conservation laws as well as our motivation for developing scalable high-performance software tools for their study. In section 3., we describe the various software packages that **PetClaw** makes use of. We briefly discuss Python and NumPy, and then introduce Clawpack and PETSc, as well as the the Python packages that provide interfaces to them. In section 4., we discuss the implementation of **PetClaw** and its performance in serial and parallel. We provide an overview of our approach to interface orthogonally into the two large, separate codebases written in Fortran and C. We also present an overview of our object-oriented design scheme, with a focus on the inheritance relationship between the **PetClaw** and **PyClaw** drivers. We present computational results demonstrating that our serial Python-based code suffers almost no degradation in performance in comparison with a pure Fortran version. Additionally, we show scalability results, for test problems involving advection and elasticity, to 16,384 cores on an IBM BlueGene/P supercomputer. Finally, in section 5. we demonstrate the usefulness of our code by studying the behavior of nonlinear elastic waves in a periodic medium.

2. NUMERICAL WAVE PROPAGATION

2.1. Hyperbolic Conservation Laws

Many important physical problems may be modelled by systems of hyperbolic conservation laws. In three dimensions, these take the form

$$q_t + f(q)_x + g(q)_y + h(q)_z = 0, \tag{1}$$

where the vector-valued function $q(x,t)$ represents the conserved quantities and $f(q), g(q), h(q)$ are fluxes. As a few important examples of (1), we mention the Euler equations of compressible fluid dynamics, the shallow water equations, Maxwell's equations describing electromagnetic waves, and the equations of elasticity.

2.2. Computational Considerations

Computational solution of nonlinear hyperbolic equations is often costly, for two reasons. First, solutions of (1) generically develop singularities (shocks) after a short time, even if the initial data are smooth. Accurate modeling of solutions with shocks generally requires the use of Riemann solvers and nonlinear limiters, which are both expensive. Thus it is essential that these routines are implemented as efficiently as possible.

Second, many hyperbolic problems exhibit a wide range of physical scales. In fluid dynamics, small scales are induced by the formation of turbulence. While turbulence models may be used to avoid resolving these scales computationally, direct numerical simulation is necessary for validation of modelling codes or to gain deeper insight. In many applications in elasticity or electromagnetics, waves propagate in a material with fine structure; e.g. metamaterials, waveguides, or even naturally occurring materials like the earth. Effective medium techniques may be used to avoid resolving the fine structure computationally, but direct simulations are necessary to validate these models and give better understanding. In section 5., we give an example of such an application where it seems that homogenization techniques are unlikely to capture the appropriate dynamics. In other applications different scales arise for various reasons. For instance, in tsunami modeling it is necessary to model entire oceans, but one is interested in predicting run-up on a scale of meters.

The appearance of widely varying spatial scales makes these problems computationally expensive since it demands the use of very fine grids. If the small scales appear only in small, localized regions, this can be mitigated by the use of adaptive mesh refinement. However, for direct simulation of waves in periodic or random media fine meshes are typically required over all or most of the computational domain. These situations demand the use of massive parallelism.

2.3. Balance Laws

Hyperbolic equations describe wave motion. Many important problems involve wave motion as well as other kinds of dynamics and may be written as balance laws:

$$q_t + f(q)_x + g(q)_y + h(q)_z = \psi(q,x,t). \qquad (2)$$

For example, the purely hyperbolic systems for fluid dynamics and electromagnetism described above are obtained by neglecting diffusive effects. These diffusive effects are significant for many important applications and must be included in ψ. Other problems in areas like combustion involve reaction terms that also may be included in ψ. The integration of these terms is often also very challenging and may require costly solvers.

3. SOFTWARE TOOLS

We have chosen to develop **PetClaw** in the high level scripting language Python, in order to simplify code development and maintenance, make maximum use of existing codes and packages in a variety of languages, and facilitate extensibility. We address the computational difficulties mentioned in the previous section by incorporating wrapped Fortran code (from Clawpack) for the expensive limiter and Riemann solver routines, and by using wrapped C/MPI code (from PETSc) for distributed parallelism.

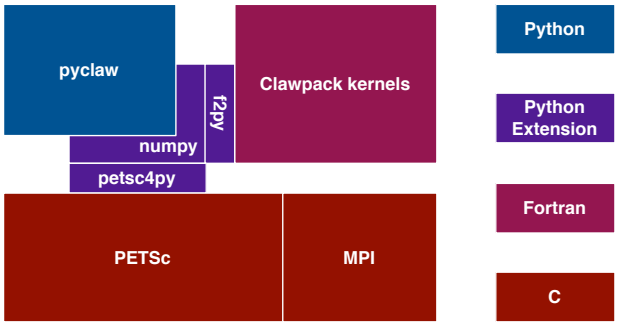


Figure 1. Modular structure of the **PetClaw** code, with a focus on the orthogonality of the Fortran kernels from the parallel decomposition through PETSc

The relation between the various software components that **PetClaw** makes use of is indicated in Figure 1. One important aspect of **PetClaw** is the extensive use of numpy to interface Python and extension module code. We describe this aspect in further detail in this section as we describe the components that constitute **PetClaw**. A second important feature of **PetClaw** is the orthogonal decomposition of the Clawpack routines for wave propagation in Fortran from PETSc objects and methods, allowing us to completely hide the details of the parallel decomposition from the the wave propagation kernels. We detail this further in section 4..

3.1. Python and numpy

Python is a widely used scripting language, and has been employed for large scale industrial (Google App Engine) and scientific [5] uses. It has a concise, readable syntax and a large support library of specialized modules, including symbolic computation [19], which is absent from scientific libraries in C and Fortran. Moreover, Python uses dynamic loading of shared libraries to incorporate modules so that its functionality can be extended smoothly at runtime. Recently, Python has become a popular choice for scientific computing [9], and many established packages have acquired Python interfaces, including PETSc [4] and Trilinos [15], which provide linear algebraic operations and solvers.

The Python **numpy** [14] module is a generic interface to raw memory arrays, such as those used in C and Fortran. It understands multidimensional arrays with arbitrary layout, yet accomodates flexible indexing syntax, such as slicing. **numpy** provides an excellent substrate for no-copy communication of data structured between Python and lower level languages, such as C and Fortran. In fact, most important structures, e.g. vectors and sparse matrices, can be directly accessed with no translation at all. Thus, a properly structured top-level Python script can flexibly drive the underlying simulation with minimal loss of efficiency.

3.2. Clawpack

Clawpack stands for "Conservation Laws Package" and was initially developed for linear and nonlinear hyperbolic systems of conservation laws, with a focus on implementing high-resolution Godunov type methods using limiters in a general framework applicable to many applications. These finite volume methods require a *Riemann solver* to resolve the jump discontinuity at the interface between two grid cells into waves propagating into the neighboring cells.

The Clawpack software (www.clawpack.org) and its extensions consisting of open source Fortran code have been freely available since 1994. More than 7,000 users have registered to download this software since it first appeared and have used it on a wide variety of problems.

The wave propagation algorithms implemented in Clawpack are described in detail in [12, 13]. The Riemann solver takes cell averages in two neighboring grid cells and determines a set of waves propagating away from the cell interface in the solution to the Riemann problem (the conservation law with piecewise constant intial data). A simple update of the cell averages based on the distance these waves propagate into the cells gives the classic Godunov method, a robust but only first-order accurate numerical method. Second order correction terms can be defined in terms of these waves and then limiters are applied to these terms in order to avoid non-physical oscillations in the solution. This is crucial near discontinuities for problems involving shock waves.

Clawpack is a very general tool in the sense that it is easily adapted to solve any hyperbolic system of conservation laws (1). The only modification required for solving a particular hyperbolic system is a change to the Riemann solver routine. In fact, Clawpack is well-suited to handle even more general hyperbolic systems that are not in conservation form or that include coefficients that vary in time and space. The application presented in Section 5. is an example of the latter. In order to apply Clawpack to any such problem, it is necessary only that the Riemann solution can be accurately approximated.

3.3. PyClaw

PyClaw is a Python package designed to facilitate the extension of the basic Clawpack routines described in Section 3.2.. This is accomplished primarily by abstracting the gridded data and evolution routines in Clawpack, defining a generic interface for them to interact through. By separating and defining interfaces for the core components of Clawpack, we create the ability to modify and extend the original routines with a minimal amount of effort. The `solver` class, for example, has been designed to be extended to new types of equations and methods. The ease at which this is accomplished is one of the main benefits of working with **PyClaw** rather than Clawpack. These ideas become more powerful when one utilizes Python packages, such as `f2py`, `fwrap`, and Cython, that facilitate the wrapping of other languages, allowing users to extend **PyClaw** using languages such as Fortran and C. All of these additions on top of the algorithms in Clawpack allow for easy extensibility and flexibility in the implementation of new applications and algorithms to Clawpack.

3.4. PETSc

The Portable Extensible Toolkit for Scientific Computing (PETSc) [1, 2], is a set of libraries for the scalable solution of scientific applications modeled by partial differential equations. PETSc employs a common workflow both to implement and test an application on a workstation initially, and to deploy it on the largest supercomputers. Scientific applications built on PETSc have been successfully scaled to billions of unknowns and hundreds of thousands of cores [17].

The core focus of PETSc has traditionally been the scalable solution of linear and nonlinear systems of equations, using sparse matrices and matrix-free methods. However, it possesses a variety of useful software constructs such as the Distributed Array (**DA**) that automatically manages the parallel mapping, distribution, and coordination of a topologically Cartesian mesh and associated fields discretized over it. A **DA** is employed to manage data parallelism for **PetClaw** simulations.

The **DA** is partitioned into rectangular blocks which tile the domain. Ghost regions are extended out into neighboring blocks by a specified number of vertices (equal to the stencil width), meaning the ghosted domains are also rectangular. By default, PETSc attempts to balance the total number of vertices in each domain, as well as minimize communication which means making domains approximately square.

Since efficient wave propagation algorithms are generally explicit, PETSc's solvers are not needed in applying **PetClaw** to purely hyperbolic problems. In the future we plan to develop **PetClaw** for application to general balance laws (2). To apply Clawpack to such problems, the user must provide a way of integrating the source term ψ. Often this is as great a challenge as solving the hyperbolic part, and may require implicit discretizations. For instance, ψ may include parabolic terms, whose efficient numerical integration relies on large linear solves. Clawpack does not provide any facility of its own for discretizing and solving such source terms. A major advantage obtained by incorporating PETSc into Clawpack is the access to PETSc's efficient scalable solvers such as its multigrid solvers, which have been shown to be effective for parabolic problems [6]. Using these solvers in **PetClaw** should be relatively straightforward since **petsc4py** (see the next section) provides an interface to them.

3.5. petsc4py

petsc4py is a set of wrappers which allow PETSc to be used directly from Python. It encompasses almost the entire functionality of PETSc, and at the same time uses native idioms to make the PETSc interface more "pythonic". Thus, **petsc4py** allows users to leverage the full power of Python while maintaining the efficiency and scalability of PETSc. Most importantly, in **PetClaw** the use of **petsc4py** makes it possible to use legacy serial Fortran kernels without modification. Only the high-level Python driver code is aware of parallel structures. Close integration with **numpy** allows array data to flow seamlessly between C and Python.

petsc4py is implemented using the Cython [18, 3] compiler. This allows for excellent efficiency, using code generation techniques to remove most of the overhead of an interpreted language. Cython also provides mechanisms targeting interoperability between **petsc4py** and other Python wrapper generators for C or Fortran (eg. SWIG and F2Py), easing the reutilization of third-party codes using PETSc.

4. IMPLEMENTATION AND PERFORMANCE

In this section, we discuss the implementation of **PetClaw** based on the tools discussed in the previous section. We then present results on the serial and parallel performance of **PetClaw**.

4.1. The PetClaw Code

It is generally accepted that an object oriented design facilitates extensibility of scientific codes. We have designed **PetClaw** with maximal code reuse from **PyClaw**. Instead of injecting parallel code throughout the original **PyClaw** source, which makes it difficult to maintain and debug, **PetClaw** inherits most of its constructs from **PyClaw** and uses polymorphism to enable parallelism by subclassing. Thus both the top- and bottom-level code components in **PetClaw** are purely serial; a very thin parallel layer (about 300 lines of code) ties them together.

PyClaw includes various superclasses to implement such things as adaptive time step control, structured grid representation, and time evolution. Most of these are used without modification in **PetClaw**. As shown in figure 2, **PetClaw** uses subclasses of a few **PyClaw** classes in order to implement the necessary parallel functionality. The Grid class is responsible for the problem description, including coordinates and auxiliary data like PDE coefficients that vary in space. It also includes a **numpy** array that contains the current solution. In the PCGrid subclass, this array is instead implemented as a PETSc vector. The ClawSolver class is responsible for evolving the solution in time. **PetClaw** subclasses ClawSolver in order to implement distributed boundary conditions, as well as communication of the maximum CFL number.

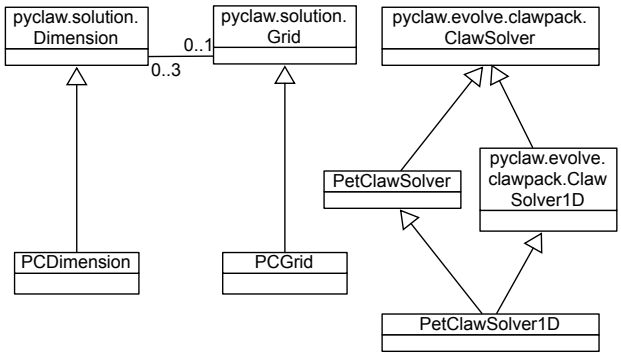


Figure 2. A Simplified class diagram of **PetClaw** classes along with their **PyClaw** superclasses

PETSc and **petsc4py** possess unified interfaces for both serial and parallel code. Python language features allow us to easily extend this functionality to the previous **PyClaw** code, which was purely serial. Thus, to adapt a **PyClaw** application to run in parallel using **PetClaw**, the only change needed is modification of a few import statements, in order to substitute the appropriate classes. For instance, in **PyClaw** the solution is stored in a **numpy** array q that is an attribute of a grid object. In **PetClaw**, we must use a PETSc class to implement q. It is therefore necessary to call **petsc4py** functions whenever q is accessed or modified. However, this bifurcation of interfaces is not very 'Pythonic', so we implement the accesses to q through a *property* function in Python. As a result, the statements

$$\texttt{grid.q} = \texttt{u}$$

and

$$\texttt{u} = \texttt{grid.q}$$

work as intended from both **PyClaw** and **PetClaw**. In **PyClaw**, these statements merely get or set values in a numpy array, whereas in **PetClaw** they request or set the contents of a PETSc vectors related to a **DA**, and communicate the new values of ghost cells (in the case of a set).

4.2. On-Core Performance Considerations and Results

One of the potential drawbacks to using Python for performance-driven applications is that it can be many times slower than equivalent Fortran or C code. The most notable reason for this is that Python is an interpreted language and incorporates dynamic type checking, which severely limits loop based algorithms without vectorization. Thus, tight loops and function calls incur a significant penalty; however, the performance penalty is small or nonexistent for logic. To maintain good efficiency, we have incorporated Fortran based Riemann solvers from Clawpack. In order to do this, we have chosen `f2py`. Although it is not the only available wrapping

package for Python, it was chosen due to the current support within the scientific computing community and its maturity as a wrapper.

We validate our approach with a performance comparison between the **PetClaw** code with python kernels (from **PyClaw**), the corresponding (Fortran-only) Clawpack code, and the hybrid **PetClaw** approach on two examples. The first example involves uniform advection, the second involves the equations of elasticity in a heterogeneous medium. The advection test is quite exacting because it involves the cheapest imaginable computational kernels, thus exposing differences in overhead costs. The elasticity example is more realistic, since it uses a more expensive Riemann solver. It also involves user-provided boundary conditions and other auxiliary problem-specific routines that are executed in Python at every time step. The convenience of being able to write these problem-specific functions in a high-level scripting language is one of the main advantages of using Python, but it is a potential performance pitfall.

Timing results are shown in Table 1. For both examples, the **PetClaw** code with python kernels is 4-5 times slower than Clawpack. By using wrapped Fortran kernels (from Clawpack), the hybrid **PetClaw** code achieves a speedup of about 3x, and is within a factor of two compared to the native Fortran code, with most of the performance difference due to object setup and deep array copies in the hybrid code. For the more realistic elasticity test, **PetClaw** is only 42% slower than hand-coded Fortran. Further analysis of the Python code might expose opportunities to reduce unnecessary copies, thus further improving the relative performance.

	Clawpack	**PetClaw** (Python)	**PetClaw** (Hybrid)
Advection	10.0	42.0	17.6
Elasticity	17.9	82.1	25.4

Table 1. Timing comparison between serial runs of **PetClaw**, Clawpack, and the hybrid **PetClaw** code for advection and elasticity. Both problems involve 10000 cells and about 10000 time steps. They were run on an Intel 3.06 GHz Intel Core 2 Duo laptop. The results are displayed in seconds.

4.3. Distributed Memory Performance Considerations and Results

PetClaw decomposes the serial computation by first partitioning cells into disjoint sets. This is accomplished through an abstract interface provided by the PETSc **DA** object, which also provides the necessary layers of "ghost" cells which border each domain. Field values are associated with each cell, and stored in PETSc **Vec** objects allocated automatically by the **DA**.

The main computational loop, in constrast, runs over cell faces. We redundantly compute flux contributions from faces shared by multiple domains. At the close of each computation step, we update the values of ghost cells from the values on the owning process. This update is also accomplished automatically by PETSc, and since it only involves nearest neighbor communication, will be scalable.

In order to ensure stability, any explicit numerical method for hyperbolic PDEs is subject to a constraint on the timestep of the form

$$\frac{\Delta t}{\Delta x} s_{\max} \leq C, \tag{3}$$

where $\Delta t, \Delta x$ are the temporal and spatial step sizes, respectively, $s_{\max}$ is the largest wave speed occurring in the problem, and C is a constant that depends on the numerical method (for Clawpack in 1D, $C = 1$). The quantity appearing on the left side of (3) is referred to as the *CFL number*. For nonlinear problems, $s_{\max}$ (and hence the CFL number) changes in time. In order to maintain an efficient timestep and to avoid violation of (3), it is necessary to calculate the local wave speeds in each cell at each time step, and perform a global reduction to find $s_{\max}$.

We conduct our computational experiments on King Abdullah University of Science and Technology's flagship supercomputer, Shaheen. Shaheen is an IBM BlueGene/P solution, comprised of 16 racks. Each rack contains 1024 quad-core PowerPC 450d compute nodes running at 850MHz. Each compute node has 4GB RAM available to its four cores.

I/O on Shaheen is provided via quad-core PowerPC 450 I/O nodes (850MHz, 4GB RAM). One quarter of the machine has 16 I/O nodes per rack, which provide a node density of 64 computational nodes (256 cores) per I/O node. The remaining three quarters of the machine has 8 I/O nodes per rack, providing a node density of 128 computational nodes (512 cores) per I/O node.

Shaheen's data storage cluster serves a collection of high-performance filesystems. It is comprised of 36 IBM System x3650s driving DCS9900 Data Storage with 2 DS5100 EXP storage drawers for metadata.

We determine the scalability of our approach with several experiments on Shaheen. We begin with a simple model, an advection problem with waves moving at constant speed. Starting with a simulation of 4096 grid cells per process on 512 processes (mapped to 512 cores on 128 nodes using Virtual Node mode), we double the number of cores while keeping the work ratio (grid cells per core) constant by doubling the total grid cells as well. We perform the first experiment with no output. To gain an appreciation for the effects of output on scalability, we perform the same weak scaling experiment, this time aggressively 'checkpointing' the solution data every 1500 time steps. Finally, to gain an understanding of the scaling behavior for a more comprehensive example, we perform a third scaling experiment modeling elastic wave propa-

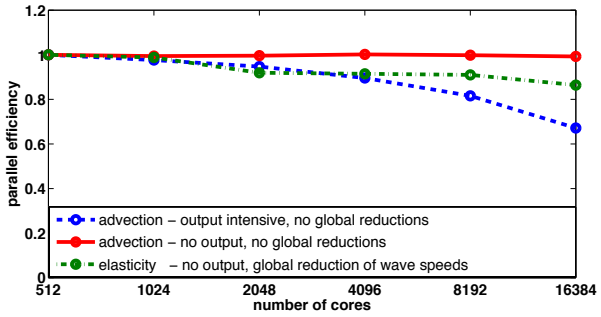


Figure 3. Weak scalability of **PetClaw** code for simple advection and elastic wave propagation in heterogeneous media.

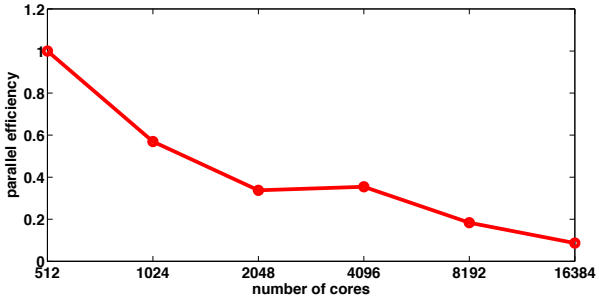
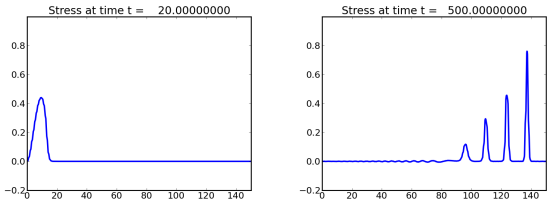


Figure 4. Catastrophic weak scaling of the dynamic loading phase during Python job startup.



(a) Initial pulse (b) Separation into solitary waves

Figure 5. Evolution of a single pulse into a solitary wave train.

gation in a periodic medium (see section 5. for more details), using the same number of grid cells per process and weak scaling and normalization procedures to calculate parallel efficiency. For nonlinear applications, in which the maximum stable timestep for the problem typically varies in time, PetClaw employs adaptive time stepping. This requires global communication of the CFL condition, which is a potential performance bottleneck for large numbers of processors.

The results of all three experiments are plotted in Figure 3. For the constant time step problem with no output, we see excellent parallel efficiency of 99% through 16,384 cores. However, the efficiency of the code when aggresively outputting the solution falls below 80% beyond 8,192 cores of Shaheen. Our effective cone of parallel efficiency when the CFL condition does not need to be globally reduced lies between the two lines and is largely dependent upon the relative speed of the I/O subsystem as well as the frequency of solution output. Introduction of global communication for adaptive time stepping causes a tolerable decrease in parallel efficiency, **PetClaw** remains above 85% parallel efficiency through 16,384 cores for the elastic wave propagation through periodic media problem.

Our software is the first production scripting language based code to run with thousands of processes on Shaheen. Consequently, we were the first users on the system to notice serious delays when loading dynamic library objects on runs with thousands of processes, with performance decreasing linearly as a function of the number of processes. Although the poor startup time is easily mitigated by long production runs, we isolate the dynamic loading phase and highlight its scalability apart from the solve phase in Figure 4. Poor dynamic library loading performance poses a hurdle for scalability of short time runs until this issue has been resolved.

5. APPLICATION TO ELASTIC WAVE PROPAGATION IN PERIODIC MEDIA

In this section, we apply the **PetClaw** code to study a novel wave phenomenon in periodic materials. LeVeque & Yong [11] discovered that nonlinear waves traveling in a periodic medium can exhibit solitary wave formation, as illustrated in Figure 5.

One remarkable aspect of this is that the formation of shocks appears to be entirely suppressed. However, this behavior depends on the particular properties of the background medium. Since steep gradients appear in the solution, and since numerical solvers necessarily smear shocks over at least a few computational cells, direct inspection does not allow any conclusions about shock formation.

The equations of nonlinear elasticity in a heterogeneous medium can be written as

$$\varepsilon_t - u_x = 0 \tag{4}$$

$$\rho(x)u_t - \sigma(\varepsilon,x)_x = 0. \tag{5}$$

Here $\varepsilon(x,t)$ is the strain, $\sigma(\varepsilon,x)$ the stress, $u(x,t)$ the velocity, and $\rho(x)$ is the density. The total energy of the system

$$\eta(u,\varepsilon,x) = \frac{1}{2}\rho(x)u^2 + \int_0^{\varepsilon} \sigma(s,x)ds. \tag{6}$$

can be used as a proxy to study shock formation, since it is conserved as long as the solution remains smooth, but decreases whenever shocks are present.

Following [11], we consider the elasticity equations (4) in a medium composed of alternating layers of two materials:

$$(\rho(x),\sigma(\varepsilon,x)) = \begin{cases} (1,\exp(\varepsilon)-1) \text{ for } 0 \le x - \lfloor x \rfloor < 1/2 \\ (Z,\exp(Z\varepsilon)-1) \text{ for } 1/2 \le x - \lfloor x \rfloor < 1 \end{cases} \tag{7}$$

Here Z plays the role of impedance for small-amplitude waves.

Results in [8] indicate that the formation of solitary waves and associated suppression of shock formation depends critically on the values of Z. There exists a relatively abrupt *phase transition* as the impedance variation increases beyond a certain threshold. Below this threshold there is significant shock formation; above the threshold there is very little. An important theoretical question is whether shock formation is suppressed completely beyond this threshold, or whether there is still formation of weak shocks; the latter behavior was observed in [16]. On the other hand, complete shock suppression would indicate the ability to transmit large-amplitude signals over arbitrary distances without information loss.

Answering this question is computationally difficult, because it requires detection of extremely small losses of energy after long simulation times. This entails the use of very fine grids. The domain must be large enough that, in a homogeneous medium, a shock wave forms and a substantial fraction of the energy is dissipated as the shock wave crosses the domain. This depends also on the shape and amplitude of the initial pulse. These considerations require a domain that is at least $O(10^3)$ layers across. We choose units so that the material layer period is 1 and the wave speeds appearing in the problem are $O(1)$. In order to reduce the numerical errors to below 10^{-8}, it turns out that a grid with $O(10^4)$ cells per material layer is needed. Hence the total number of grid cells required is $O(10^7)$; the number of time steps to be taken is of the same magnitude. The largest runs here used over 7 million cells and over 15 million timesteps.

On a 2.66 GHz Intel Xeon processor, a single timestep on this grid takes 2 seconds. Hence the full simulation would require approximately 0.95 years. Instead, this computation was completed in about 15 hours on 8192 cores of Shaheen. In order to understand this behavior more deeply we plan to conduct detailed parameter studies, involving hundreds of simulations. Clearly, this would be impossible on a serial machine.

In order to solve this system in **PetClaw** we use the f-wave approximate Riemann solver for nonlinear elasticity developed in [10]. The total energy loss as a function of the medium parameter Z is plotted in figure 6. These computations bound the energy loss due to shock formation for large Z as no greater than 4.4×10^{-9}. For comparison, a second curve is plotted showing results obtained in serial on a workstation in about the same amount of time. For those runs, the energy

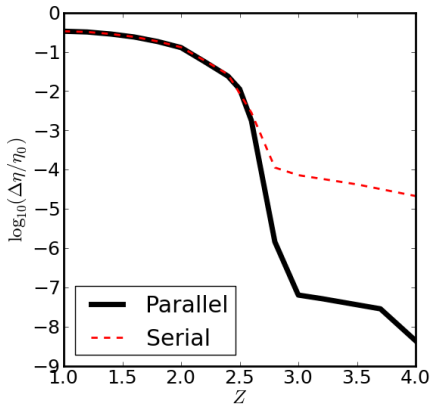


Figure 6. Relative energy loss versus impedance contrast for a nonlinear pulse in a periodic medium. For large enough impedance Z, energy is conserved up to numerical errors.

loss due to numerical errors is about 10^{-4}. By using the parallel PetClaw code, we have obtained results (on a grid 64 times finer) that are more accurate by 4 orders of magnitude and demonstrate clearly the existence of a "phase change" with respect to shock formation. In the near future we plan to incorporate higher order accurate methods into **PetClaw**, which should enable us to reduce this bound to near machine precision.

6. CONCLUSION

Since it builds on and maintains the interface of a widely-used serial code, **PetClaw** can leverage the investment of the existing community, and promises to have a strong impact on applications involving hyperbolic PDEs. It employs a very general framework that requires modification of only a single component, the Riemann solver, in order to solve a wide range of problems. On the other hand, it avoids the need for new users to write Fortran code in most cases, since existing Fortran Riemann solvers can be employed in very many applications, and the remaining code can be written in easily in C or Python. In much the same way as PETSc, **PetClaw** removes the need for a user to be familiar with MPI before running large computations scalably on supercomputing platforms. Moreover, inclusion of accurate, scalable hyperbolic solves inside larger simulation codes now becomes quite easy using Python scripting.

The next release of **PetClaw** will incorporate both 2D and 3D solvers. This will be relatively straightforward, since both Clawpack and PETSc are already 3D-capable. In collaboration with the Argonne Leadership Computing Facility, development and incorporation of rapid dynamic loading strategies for thousands of processes will enable the full Python application to scale to the entire BG/Q machine. Inclusion of WENO

reconstruction and Runge-Kutta time integration, also leveraging existing Fortran code [7], will enable high order accurate solvers.

In the longer term, **PetClaw** will include both limiting and Riemann solves for many-core architectures, using CUDA and OpenCL, to enable scaling on hybrid architectures. We will incorporate PETSc's efficient solvers for parabolic and other stiff source terms to handle general balance laws. We also plan to develop an adaptive mesh refinement capability similar to that in AMRCLAW.

The interested reader may obtain code and instructions for reproducing the tests in this paper from `http://web.kaust.edu.sa/faculty/davidketcheson/PetClaw_HPC_paper.html`

REFERENCES

[1] Satish Balay, Kris Buschelman, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.0.0, Argonne National Laboratory, 2009.

[2] Satish Balay, Kris Buschelman, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc Web page. `http://www.mcs.anl.gov/petsc`, 2010.

[3] Stefan Behnel, Robert Bradshaw, Craig Citro, Lisandro Dalcin, Dag Sverre Seljebotn, and Kurt Smith. Cython: The best of both worlds. *Computing in Science and Engineering*, September 2010.

[4] Lisandro Dalcin. petsc4py web page. `http://petsc4py.googlecode.com`, 2010.

[5] J Enkovaara, C Rostgaard, J J Mortensen, J Chen, M Dułak, L Ferrighi, J Gavnholt, C Glinsvad, V Haikola, H A Hansen, H H Kristoffersen, M Kuisma, A H Larsen, L Lehtovaara, M Ljungberg, O Lopez-Acevedo, P G Moses, J Ojanen, T Olsen, V Petzold, N A Romero, J Stausholm-Møller, M Strange, G A Tritsaris, M Vanin, M Walter, B Hammer, H Häkkinen, G K H Madsen, R M Nieminen, J K Nørskov, M Puska, T T Rantala, J Schiøtz, K S Thygesen, and K W Jacobsen. Electronic structure calculations with gpaw: a real-space implementation of the projector augmented-wave method. *Journal of Physics: Condensed Matter*, 22(25):253202, 2010.

[6] Michael Holst and Faisal Saied. Parallel performance of some multigrid solvers for three-dimensional parabolic equations. Technical report, UIUC, 1991.

[7] David I Ketcheson and Randall J LeVeque. WENOCLAW: A higher order wave propagation method. In *Hyperbolic Problems: Theory, Numerics, Applications: Proceedings of the Eleventh International Conference on Hyperbolic Problems*, page 1123, Berlin, 2008.

[8] D.I. Ketcheson and R.J. LeVeque. Suppression of Shock Formation by Periodic Materials. In preparation, 2010.

[9] Hans Petter Langtangen, editor. *Python Scripting for Computational Science*, volume 3 of *Texts in Computational Science and Engineering*. Springer, 3rd edition, 2008.

[10] R J LeVeque. Finite-volume methods for non-linear elasticity in heterogeneous media. *IJNMF*, 40:93–104, 2002.

[11] R J LeVeque and D H Yong. Solitary waves in layered nonlinear media. *SIAM Journal of Applied Mathematics*, 63:1539–1560, 2003.

[12] Randall J LeVeque. Wave Propagation Algorithms for Multidimensional Hyperbolic Systems. *Journal of Computational Physics*, 131:327–353, 1997.

[13] R.J. LeVeque. *Finite volume methods for hyperbolic problems*. Cambridge University Press, Cambridge, 2002.

[14] Travis E. Oliphant. *A Guide to NumPy*. Trelgol, 2006.

[15] M. Sala, W. Spotz, and M. Heroux. PyTrilinos: High-performance distributed-memory solvers for Python. *ACM Transactions on Mathematical Software (TOMS)*, 34, March 2008.

[16] G. Simpson and M. I. Weinstein. Coherent Structures and Carrier Shocks in the Nonlinear Periodic Maxwell Equations. *ArXiv e-prints*, September 2010.

[17] M. A. Smith, C. Rabiti, D. Kaushik, B. Smith, W. S. Yang, and G. Palmiotti. Fast reactor core simulations using the unic code. In *Proceedings of the International Conference on the Physics of Reactors, Nuclear Power: A Sustainable Resource*, 2008.

[18] Cython Team. Cython: C–Extensions for Python, 2010. `http://www.cython.org`.

[19] Ondřej Čertík. sympy web page. `http://www.sympy.org`, 2010.