Application of the Proteus Toolkit to Marine, Coastal, and Riverine Engineering Problems

Chris Kees, US Army ERDC-CHL
Giovanni Cozzuto, HR Wallingford
Aggelos Dimakopoulos, HR Wallingford
Tristan de Lataillade, Universite of Edinburgh

Key words: Computational Fluid Dynamics, Finite Element Method, Fluid-Structure Interaction, Breakwaters, Waves

Introduction

Three-dimensional computational modeling of wave and current interaction with dynamic structures is becoming a reality for engineering analyses that have traditionally relied heavily on physical testing. While application of computational fluid dynamics is already commonplace in many other areas of engineering, the difficulties posed by complex free surface dynamics and deforming geometries have constrained the size of the community employing 3D free-surface fluid-structure interaction tools and limited widespread adoption. While advances in computational methods for this class of problems have been of primary importance in achieving a robust, accurate, and efficient modeling capability, better algorithms alone are insufficient for realizing the full impact of computational fluid-structure interaction analyses in coastal and hydraulic engineering. Instead, there are several gaps, including the problem-implementation gap, which limit the impact of computational modeling advances.

In this work we present experience developing, applying, and extending the Proteus toolkit to applications in marine, coastal, and hydraulic engineering. Proteus is an open source middleware toolkit for computational methods and simulation that has served as a platform for research in numerical methods and continuum modeling [2]. In expanding to support more applications, however, it has been necessary to address several gaps:

- Specification of domain geometry, boundary conditions, and wave climate is too complex when implemented through application programming interfaces designed for generic partial differential equations.
- Software distribution and build issues become a major barrier to expanding past the expert developer base.
- The needs of real applications often result in custom code that can reverse efficiency gains obtained through research and optimization.

Domain-Specific Interfaces

Proteus was initially designed for supporting multi-physics and multi-numerics modeling for equations arising in continuum mechanics. For this reason it contains high-level abstractions representing partial differential equations, domains, boundary conditions, and initial conditions. The equations are abstract high-order transport equations since such equations arise natural as the balance equations for mass, momentum, and energy in continuum mechanics. These “physics” abstractions are independent of any choices on the numerical methods (“numerics”) used to solve the problem. Models of free-surface fluid-structure interaction are often represented by a coupled system of ten or more equations and solution components. Using the abstract interface to boundary conditions results in a combinatorial explosion of ways to represent boundary conditions, many of which are non-physical or ill-posed. To remedy this situation, we implemented a layer of higher-level boundary conditions that map directly from the physical boundary condition types to their optimal mathematical representations. In a similar fashion we developed a set of spatial domain types that provide basic composition of subdomains (Boolean operations) while naturally allowing specification of boundary conditions on the elementary subdomains. This allows users to intuitively build more complex domain geometries from simpler ones without creating problems for later mesh generation and adaption stages.

Pragmatic Functional Package Management

Any reasonably complex software tool faces decisions about whether to “roll your own” or reuse existing libraries. One problem that continues to plague the latter approach is the “dependency hell” resulting from the need to track, build, and port multiple external software projects. For example, a recent coastal engineering demonstration application has package dependency represented in figure 1. A novel functional approach to
managing complex package dependencies was proposed in [4] and implemented on unix systems as the Nix package manager. Unfortunately, on High Performance Computing systems one frequently encounters the need to build with some vendor-supplied libraries that break the functional paradigm. To support this slightly imperfect functional approach we developed the HashDist tool in collaboration with several other research organizations [1]. HashDist uses a YAML mini-language to describe both packages and collections of packages called profiles. Using version control on the set of package and profile specs results in a precise, reproducible specification of sources and build instructions for assembling complete stacks. Recently we have worked with Kitware, Inc. and Anaconda, Inc. to improve integration with the CMake build system and conda package manager and used Docker containers to deploy Proteus in a variety of contexts.

Figure 1: Dependency graph for the packages required in a recent coastal modeling application.

Spiral Development with Python and C++

Another common tension in software applications involving complex models arises due to the appropriate use of different programming languages. While general systems program languages like C++ now offer efficiency along with versatility, interpreted languages such as Python or specialized languages such as Matlab offer higher productivity for many tasks. In the pioneering work of [5], Python was suggested as a complement to either Fortran, C, or C++. In the decades since, tools for working in C++ and Python have dramatically improved. While new languages on the horizon, such as Julia [3], may offer the advantages of interpreted languages while providing the performance of compiled languages, Proteus continues at present with high-level API’s and algorithms in Python with translation and design in C++ as needed to obtain optimal performance. Several attempts have been made to automate translation to C++ in some of these instances by developing embedded Domain Specific Languages in Python, but these attempts have not been succesful for production applications.

References