Parallel Computational Methods and Simulation for Coastal and Hydraulic Applications Using the Proteus Toolkit

C. E. Kees and M. W. Farthing
Coastal and Hydraulics Laboratory
US Army Engineer Research and Development Center
3909 Halls Ferry Road, Vicksburg, MS 39180-6133, USA
Email: chris.kees@us.army.mil & matthew.w.farthing@usace.army.mil

***Abstract*—The Proteus toolkit evolved to support research on new models for coastal and hydraulic processes and improvements in numerical methods. The models considered include multiphase flow in porous media, shallow water flow, turbulent free surface flow, and flow-driven processes such as sediment and species transport. Python was used for implementing high-level class hierarchies and prototyping new algorithms, while performance critical sections were optimized using compiled languages. In this paper we present an overview of the toolkit design, some examples, and open issues.**

***Keywords*-partial differential equations; finite element methods; hydrodynamics; parallel algorithms;**

I. Introduction

Proteus (http://proteus.usace.army.mil) is a Python package for rapidly developing computer models and numerical methods. The focus is on continuum mechanical models of hydrodynamics and transport processes in complex natural and engineered settings. Generally such models reduce to initial-boundary value problems for nonlinear partial differential equations on geometrically complex domains. Models of this type arise frequently in the course of civilian and military engineering projects of the U.S. Army Corps of Engineers, and the Proteus package was developed as part of the authors' research and development projects over the past several years at the U.S. Army Engineer Research and Develoment Center (ERDC) [1]–[7].

The Proteus package contains a collection of modules implemented in C, C++, Fortran, and Python. The fundamental design principle of Proteus is to loosely couple the implementation of the physics, that is the initial-boundary value problems for systems of partial differential equations, and the numerical methods, which at present includes, but is not limited to, a wide range of finite element methods. The objective is to make it easy for scientists to implement new problems (equation sets, initial and boundary conditions, and geometry) without having to commit to a specific mesh, discretization, or solvers. Likewise numerical analysts can develop numerical methods without committing to a small set of simplified test problems and instead can access large collections of relevant and challenging applications for testing and guiding their numerical analysis research.

As a side effect of this design choice, the framework also allows for unifying existing legacy codes under a single framework where they can share modules for common i/o, visualization, and geometry processing tasks. Implementing these tasks in a "stove-piped" approach consumes limited development resources and frequently leads to unhealthy competition for funding among otherwise complementary research groups within and across organizations.

The objectives of this paper are to 1) give a high-level overview of Proteus, 2) present some simple examples that illuminate the primary software design goals, and 3) show how this approaches yields massively parallel programs with minimal parallel programming knowledge required of model and method developers.

II. Overview

There has been intense interest in the development of Problem Solving Environments (PSEs) for continuum modeling with PDEs during the last three decades, see for example [8]–[11]. Part of this interest was due to the perceived generality of certain numerical methods, particularly finite element methods (FEMs). As numerical methods for PDEs have expanded into solving real-world problems, the idealistic notion of an expert system for generic PDEs or even a limited class of first or second order balance laws, has been tempered by the realization that good models of physical systems are require highly specialized methods that explicitly take the physics of the application into account. Often this specialized knowledge is focused on the representation of so-called "sub-grid-scale effects". Examples of physics requiring considerations of sub-grid effects include turbulence in open channel flows, heterogeneity in subsurface porous media, and dispersion in transport phenomena. Our motivation for designing a system that loosely couples physics and numerics is not, therefore, to build a kind of expert system that solves all problems with an optimal method. Instead, through our loosely coupled physics and numerics, we hope to achieve a toolkit that will allow experts flexibility in addressing and balancing the particular objectives of their modeling and simulation projects.

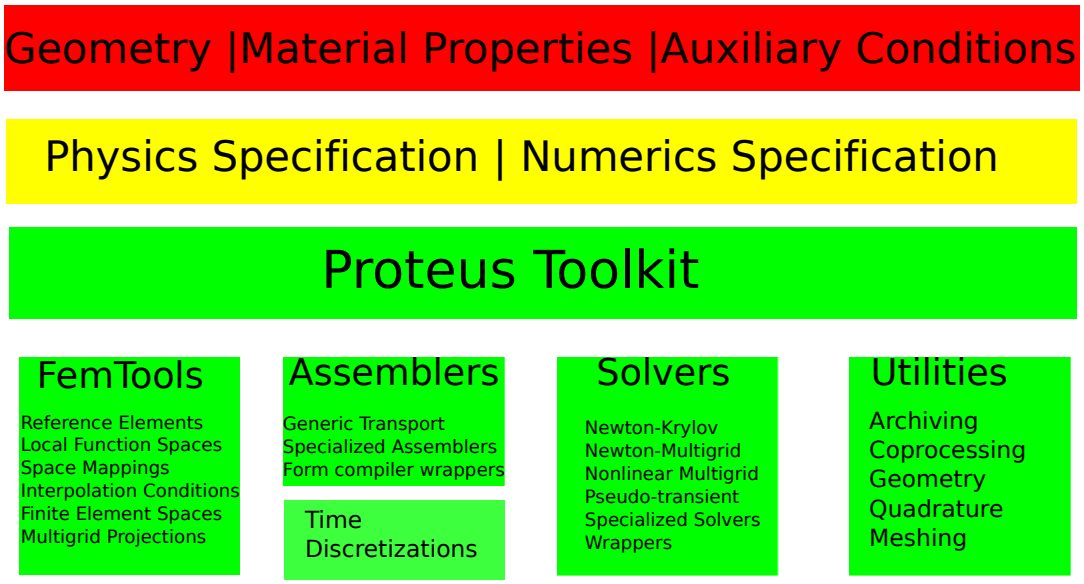

Figure 1. Modular structure of Proteus

The Proteus toolkit contains many examples of physics (equation sets), numerics (finite element methods), and test problems collected over the past five years of our research and development projects considering both mathematical model and numerical methods development. Model equations implemented include:

- 2D and 3D incompressible Navier-Stokes (Unsteady/Steady, LES, RANS, VANS)
- 2D diffusive wave (overland flow)
- 2D shallow water
- 2D and 3D two-phase incompressible, immiscible flow (hybrid VOF/level set formulation with LES, etc.)
- 2D and 3D saturated groundwater
- 2D and 3D Richards' equation (variably saturated groundwater, various constitutive models)
- 2D and 3D two-phase flow in porous media (continuum mixture formulation, incompressible or compressible)
- 2D and 3D density-dependent groundwater flow and salinity transport
- 2D and 3D eikonal equation (signed distance calculations)
- 2D and 3D linear elasticity
- 3D elastoplastic deformation (levee stability, Mohr-Coulomb material)
- 2D and 3D 6DOF solid/air/water interaction
- 1D,2D,and 3D Poisson, Burgers, linear/nonlinear ADRE, Stokes, etc.

Numerical methods implemented include:

- Continuous linear and quadratic polynomial spaces (C^0P^1 and C^0P^2) on simplicial elements (intervals, triangles, tetrahedra) with nodal (Lagrange) basis
- Continuous tensor product spaces (C^0Q^k) on hexahedra with nodal basis
- Discontinuous complete polynomial spaces ($C^{-1}P^k$) on simplicial elements with monomial basis
- P^1 non-conforming simplicial elements (equivalent to Raviart-Thomas mixed element)
- Eulerian-Lagrangian Localized Adjoint Methods (ELLAMs) for advection-dominated processes
- Locally discontinuous Galerkin mixed elements with static condensation
- SIPG/NIPG/IIPG primal discontinuous elements
- Residual-based variational multiscale methods (RBVMS)
- Analytical Riemann solvers (numerical fluxes) for linear advection, two-phase flow in porous media, and shallow water
- Approximate Riemann solvers: Harten-Lax-van Leer (SWE), Rusanov (two-phase flow), Cheng-Shu (Hamilton-Jacobi)
- Velocity post-processing to enforce element-wise (local) conservation

We developed a wide range of applications, benchmarks, and verification problems, including:

- Dam break experiments
- Marin free surface flow/object experiment
- Wigley hull tow tank experiment
- Beach erosion board
- Flow around a cylinder
- Driven cavity
- Rotating Gaussian
- Advection in a vortex
- Porous media and slope stability
- Poisseulle, Couette, and Decay of Vortex (low RE

analytical solutions) 2D and 3D

III. Physics Interface

In the previous section we gave some idea of what Proteus can do. In this section we provide more detail on how model and method developers can use Python to extend Proteus to their particular research area. First we provide some motivation of the interface design.

A. Some Characteristics of Popular Math Software

Surveying popular mathematical software, we concluded that it often combines the following characteristics:

- It is directed at an abstract formulation covering an important class of problems. For example, linear algebraic systems $A\mathbf{x} = \mathbf{b}$, $\mathbf{F}(t, \mathbf{y}, \mathbf{y}') = 0$.
- It has multiple layers of interfaces. For example, the popular LAPACK library has two interfaces for solving the same general banded system [12]: `dgbsv` (simple interface), `dgbtrf` + `dgbtrs` (computational interface)
- It uses robust and accurate numerics: LU with partial pivoting, BDF methods.
- It provides a separation between problem/data description and numerics. For example, the popular PETSc toolkit for parallel methods choose to implement separate class hierarchies for matrices and Krylov solvers instead of adding solvers as methods of matrix data types [13]–[15]: (PetscMat,PetscKSP).

We have tried to employ these in our design, in particular we have chosen to focus on a subset of continuum mechanical balance laws and provide three successively more efficient but more involved interfaces to representing physics.

B. Second Order Nonlinear, Heterogeneous Transport Systems

Our target problems are systems of nonlinear equations governing the transport of an abstract vector of components $u_j, j = 1, \ldots, n_c$:

$$\frac{\partial m^i}{\partial t} + \nabla \cdot \left(\mathbf{f}^i - \sum_k^{n_c} \boldsymbol{a}^{ik} \nabla \phi^k \right) + r^i + h^i(\nabla u) = 0 \quad (1)$$

where $i = 1, \ldots n_c$. We will adopt the following terminology for the differential operators in this equation:

mass	advection	diffusion	reaction	hamiltonian
m^i	$\mathbf{f}^i$	$\boldsymbol{a}^{ik}$	r^i	h^i

where ϕ^k, which we label the potential, is also allowed to be nonlinear in the solution. Our approach to implementing a specific form of this equation is simple: First indicate the form of the nonzero coefficients in the equation using a Python dictionary and second implement a function for setting the value the coefficients as a function of time, space, and the unknown solution. This process is most easily described by an example.

1) Linear Advection Diffusion Example: The initial-boundary value problem is: For $(t, x, y) \in [0, T] \times [0, 1] \times [0, 1]$ find u such that

$$\begin{aligned} (Mu)_t + \nabla \cdot [\mathbf{B}u - \boldsymbol{A}\nabla u] &= 0 \\ u(0, x, y) &= 0 \\ u(t, x, 0) &= u(t, 0, y) = 1 \\ u(t, x, 1) &= u(t, 1, x) = 0 \\ M &= 1 \\ \mathbf{B} &= (2, 1) \\ \boldsymbol{A} &= 0.001\boldsymbol{I} \end{aligned}$$

2) Implementing the abstract physics: This model equation governs, for example, transport of a contaminant in a waterway. To describe this equation we derive a class from the base class `TC_Base` in the `TransportCoefficients` module. Figure 2 shows how the differential operators are "turned on" through a dictionary and then the coefficients of theses operators are defined by the `evaluateCoefficients` function.

3) Implementing the site-specific physics: Next we need to specify the initial-boundary conditions of the problem at hand. Figure 3 provides the code that instantiates the ADR object, and sets particular coefficient values and boundary conditions.

4) Implementing the site-specific numerics: The initial-boundary value problem for the advection-diffusion equation that we are solving is essentially a singularly perturbed problem due to the small diffusion coefficient. For that reason we apply the so-called Algebraic-Sub-Grid-Scale stabilization to the standard Galerkin method. Figure 4 provides the code that sets up the required numerical methods to solve the problem.

C. A multi-physics example

Before proceeding to the high-performance computing aspects of Proteus, we add a more complex example that shows how multiple operators and equation sets can be coupled weakly through split operator methods. The basic idea is to form a list of complete computational models and provide an approach for lagging and/or iterating to achieve the desired multi-physics model. Figure 5 provides and example of an split operator module for the solution of two-phase flow.

D. Enabling parallelism

Parallelizing finite element codes is a non-trivial aspect of their implementation due to the spatial coupling that is inherent in PDEs. In Proteus we implement parallelism using the Message Passing Interface (MPI) and the Portable Extensible Toolkit for Scientific Computation (PETSc) as well as the Python wrapper packages mpi4py and petsc4py [15], [16]. Using these tools we are able to insulate the model developers from the large majority of the parallel

```python
from proteus.TransportCoefficients import *

class LAD(TC_base):
    """
    The coefficients of the linear advection-diffusion equation
    """
    def __init__(self,M,A,B):
        TC_base.__init__(self,
                         nc=1, #number of components
                         variableNames=['u'],
                         mass      = {0:{0:'linear'}},
                         advection = {0:{0:'linear'}},
                         diffusion = {0:{0:{0:'constant'}}},
                         potential = {0:{0:'u'}},
                         reaction  = {0:{0:'linear'}})
        self.M=M;
        self.A=A;
        self.B=B;

    def evaluate(self,t,c):
        c[('m',0)][:]          = self.M*c[('u',0)]
        c[('dm',0,0)][:]       = self.M
        c[('f',0)][...,0]      = self.B[0]*c[('u',0)]
        c[('f',0)][...,1]      = self.B[1]*c[('u',0)]
        c[('df',0,0)][...,0]   = self.B[0]
        c[('df',0,0)][...,1]   = self.B[1]
        c[('a',0,0)][...,0,0] = self.A[0][0]
        c[('a',0,0)][...,1,1] = self.A[1][1]
```

Figure 2. The `adr.py` module for defining the linear advection-diffusion equation. First set up dictionaries defining the non-zero structure of the PDE, and then define an `evaluate` function to set the numerical values as a function of time, space, and the unknown solution.

```python
from proteus import *
from proteus.default_p import *
from adr import *

name = "ladr_2d"
nd = 2; #Two dimensions
L=(1.0,1.0,1.0);
T=1.0

coefficients=LAD(M=1.0,
                 A=[[0.001,0.0],
                    [0.0,0.001]],
                 B=[2.0,1.0])

def getDBC(x,flag):
    if x[0] == 0.0 or x[1] == 0.0:
        return lambda x,t: 1.0
    elif x[0] == 1.0 or x[1] == 1.0:
        return lambda x,t: 0.0

dirichletConditions = {0:getDBC}
advectiveFluxBoundaryConditions = {}
diffusiveFluxBoundaryConditions = {0:{}}

class IC:
    def __init__(self):
        pass
    def uOfXT(self,x,t):
        if x[0] <= 0.0 or x[1] <= 0.0:
            return 1.0
        else:
            return 0.0

initialConditions  = {0:IC()}
```

Figure 3. The `ladr_2d_p.py` "p-file" for the site-specific physics

```
from proteus import *
from proteus.default_n import *
from ladr_2d_p import *
timeIntegration = BackwardEuler_cfl
stepController = Min_dt_cfl_controller
runCFL=1.0
femSpaces = {0:C0_AffineLinearOnSimplexWithNodalBasis}
elementQuadrature = SimplexGaussQuadrature(nd,3)
elementBoundaryQuadrature = SimplexGaussQuadrature(nd-1,3)
subgridError = AdvectionDiffusionReaction_ASGS(coefficients,nd,lag=False)
shockCapturing = ResGradQuad_SC(coefficients,nd,
                                shockCapturingFactor=0.99,
                                lag=True)
numericalFluxType = Advection_DiagonalUpwind_Diffusion_SIPG_exterior
nnx=41; nny=41
tnList=[float(i)/40.0 for i in range(11)]
matrix = SparseMatrix
multilevelLinearSolver = KSP_petsc4py
linearSmoother = Jacobi
l_atol_res = 1.0e-8
parallelPartitioningType = MeshParallelPartitioningTypes.node
nLayersOfOverlapForParallel = 0
```

Figure 4. The `ladr_2d_n.py` "n-file" for the site-specific numerics. The options set in this module guide how the abstract physics converted to a finite dimensional algebra problem and solved over a set of time intervals.

```
from proteus.default_so import *
pnList = [("curvature_2d_p" ,
           "curvature_2d_n"),
          ("twp_navier_stokes_2d_p",
           "twp_navier_stokes_2d_n"),
          ("ls_2d_p",
           "ls_2d_n"),
          ("vof_2d_p",
           "vof_2d_n"),
          ("redist_2d_p" ,
           "redist_2d_n"),
          ("ls_consrv_2d_p" ,
           "ls_consrv_2d_n")]
name = "twp_navier_stokes_2d"
systemStepControllerType = \
    Sequential_MinAdaptiveModelStep
tnList = [0.0,10.0]
```

Figure 5. A multi-physics example that combines incompressible flow and transport equations to approximately model incompressible air/water flow

programming. To illustrate a full parallel model using the advection-diffusion example above, we present a complete IPython notebook in figure 6.

IV. Conclusion

In this paper we have demonstrated how loosely coupling the physics from the numerics benefits users an developers in several important ways. First, it allows libraries of model equations and test problems to be accumulated without committing to a particular class of numerical method. This is critically important for large organizations that depend on modeling because it allows for continuous input and improvement in both numerical methods and their implementation. Second, this loose coupling between physics and numerics also yields simulators that can transition to parallel architectures with minimal effort. While the current implementation of the toolkit is fully functional, we intent to pursue more elegant and concise representations of the physics through the use of symbolic mathematics, higher performance and reduced memory usage through code generation (as well as better algorithms), and significant improvements in the interactivity through more advanced web and network transport technologies.

Another extremely important aspect of future work is benchmarking, comparison, and integration with other problem solving environments. Proteus evolved to meet the distinct needs of computational mechanics at the US Army Engineer Research and Development Center. These needs include support for spatial heterogeneity and temporal variation in PDE coefficients and boundary conditions, hyperbolic or nearly hyperbolic equations, geometrically messy domains, and the need for simulation across large time and space scales. These needs make it critical for proteus to support parallelism, unstructured meshes, and spatially distributed coefficients. While other packages for solving PDE's in Python do exist, it was not within the scope of this paper to make careful comparisons with these other packages. Instead we hope the overview of Proteus will spur further discussion on a common API for PDE-based models and the continuing need for community benchmark problems.

Acknowledgment

Permission was granted by the Chief of Engineers to publish this information.

References

[1] C. Kees, M. Farthing, S. Howington, E. Jenkins, and C. Kelley, "Nonlinear multilevel iterative methods for multiscale models of air/water flow in porous media," in *Proceedings of the XVI International Conference on Computational Methods in Water Resources*, Copenhagen, Denmark, 2006.

```
from IPython.parallel import Client
c = Client()
view = c[:]
%load_ext parallelmagic
view.activate()
%autopx #switch to MPI tasks
from proteus.iproteus import *
from proteus import default_n,default_s,default_so
import ladr_2d_p,ladr_2d_n
pList = [ladr_2d_p]
nList = [ladr_2d_n]
so = default_so
so.name = pList[0].name = "ladr_2d"
so.sList=[default_s]
so.tnList = ladr_2d_n.tnList
nList[0].multilevelLinearSolver=default_n.KSP_petsc4py
ns = NumericalSolution.NS_base(so,pList,nList,so.sList,opts)
ns.calculateSolution('run1')
x = ns.modelList[0].levelModelList[-1].mesh.nodeArray[:,0]
y = ns.modelList[0].levelModelList[-1].mesh.nodeArray[:,1]
triangles = ns.modelList[0].levelModelList[-1].mesh.elementNodesArray
u = ns.modelList[0].levelModelList[-1].u[0].dof
%autopx #switch back to client task
x = numpy.concatenate(view['x'])
y = numpy.concatenate(view['y'])
u = numpy.concatenate(view['u'])
n0=0
n1=n0+len(view['x'][0])
n2=n1+len(view['x'][1])
n3=n2+len(view['x'][2])
triangles = numpy.concatenate((view['triangles'][0]+n0,
                               view['triangles'][1]+n1,
                               view['triangles'][2]+n2,
                               view['triangles'][3]+n3))
tricontourf(x,y,triangles,u)
```

Figure 6. Example of interactive parallel model using IPython

[2] C. Kees, M. Farthing, and C. Dawson, "Locally conservative, stabilized finite element methods for variably saturated flow," *Computer Methods in Applied Mechanics and Engineering*, vol. 197, pp. 4610–4625, 2008.

[3] C. E. Kees, M. W. Farthing, R. Berger, and T. Lackey, "A review of methods for moving boundary problems," U.S. Army Engineer Research and Development Center, Coastal and Hydraulics Laboratory, Technical Report TR-09-10, 2009.

[4] M. W. Farthing and C. E. Kees, "Evaluating finite element methods for the level set equation," U.S. Army Engineer Research and Development Center, Coastal and Hydraulics Laboratory, Technical Report TR-09-11, 2009.

[5] C. E. Kees and M. W. Farthing, "Conservative level-set methods for two-phase flow on unstructured meshes," in *International Conference on Computational Methods in Marine Engineering MARINE 2009*, T. Kvamsdal, B. Pettersen, P. Bergan, E. O. nate, and J. Garcia, Eds. CIMNE, Barcelona, 2009.

[6] M. W. Farthing and C. E. Kees, "Locally conservative, stabilized finite element methods for a class of variable coefficient navier-stokes equations," U.S. Army Engineer Research and Development Center, Coastal and Hydraulics Laboratory, Technical Report TR-09-12, 2009.

[7] C. E. Kees, I. Akkerman, M. W. Farthing, and Y. Bazilevs, "A conservative level set method suitable for variable-order approximations and unstructured meshes," *Journal of Computational Physics*, vol. 230, no. 12, 2011.

[8] T. Zimmermann, Y. Dubois-Pélerin, and P. Bomme, "Object-oriented finite element programming: I. governing principles," *Computer Methods in Applied Mechanics and Engineering*, 1992.

[9] A. M. Bruaset and H. P. Langtangen, "Object-oriented design of preconditioned iterative methods in diffpack," *ACM Transactions on Mathematical Software*, vol. 23, no. 1, pp. 50–80, 1997.

[10] R. Kirby and A. Logg, "A compiler for variational forms," *ACM Transactions on Mathematical Software*, vol. 32, no. 3, 2006.

[11] A. Logg and G. N. Wells, "Dolfin: Automated finite element computing," *ACM Transactions on Mathematical Software*, vol. 37, no. 2, 2010.

[12] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, 3rd ed. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1999.

[13] S. Balay, J. Brown, K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, "PETSc Web page," 2011, http://www.mcs.anl.gov/petsc.

[14] 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," Argonne National Laboratory, Tech. Rep. ANL-95/11 - Revision 3.1, 2010.

[15] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith, "Efficient management of parallelism in object oriented numerical software libraries," in *Modern Software Tools in Scientific Computing*, E. Arge, A. M. Bruaset, and H. P. Langtangen, Eds. Birkhäuser Press, 1997, pp. 163–202.

[16] L. Dalcin, R. Paz, P. Kler, and A. Cosimo, "Parallel distributed computing using python," *Advances in Water Resources*, vol. 34, no. 9, pp. 1124–1139, 2011.