PECOS

Predictive Engineering and Computational Sciences

libMesh: Past, Present, and Future

Roy H. Stogner

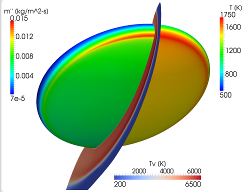
The University of Texas at Austin

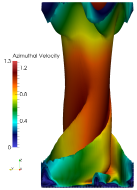
February 27, 2017

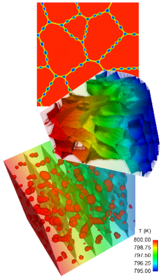
libMesh Community

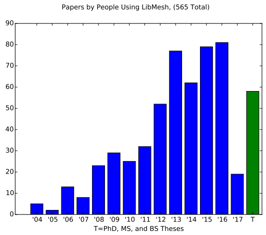
Scope

- Free, Open source
  - LGPL2 for core
- 45 Ph.D. theses, 507 papers (81 in 2015)
- ~ 10 current developers
- $110 - 240$ current users?









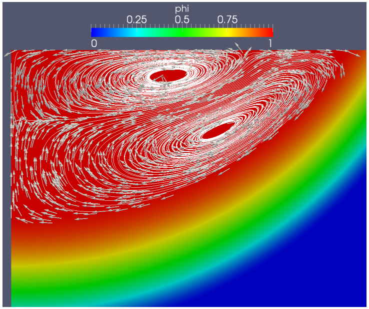
Challenges

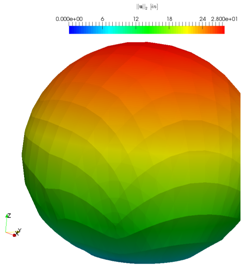
- Radically different application types
- Widely dispersed core developers
  - INL, UT-Austin, U.Buffalo, JSC, MIT, Harvard, Argonne
- OSS, commercial, private applications

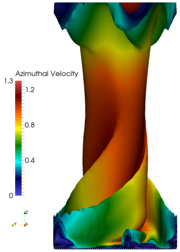
GRINS

https://github.com/grinsfem/grins

- Multiphysics FEM platform built on `libMesh`
- Modular structure for “Physics”, solvers, QoIs, etc.
- Key feature: automatically enabled discrete adjoints (AMR, sensitivities)

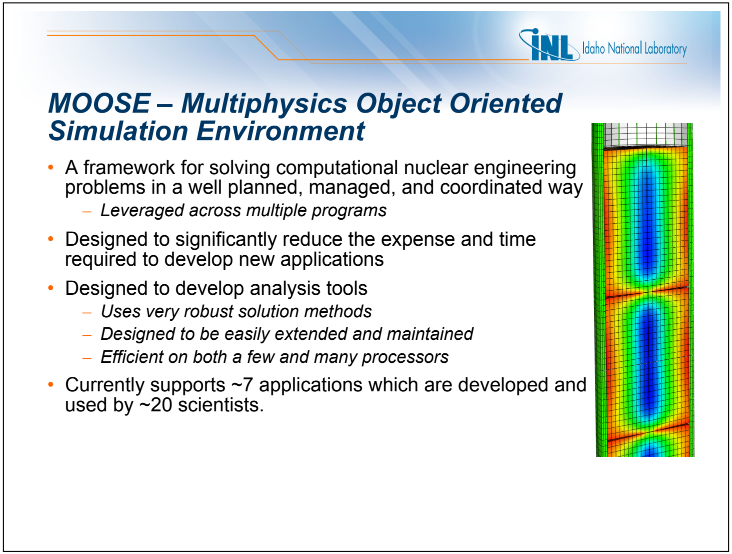






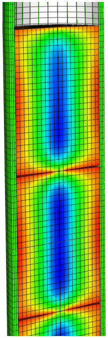
Courtesy Nick Mal

The MOOSE Framework - Gaston et al., INL

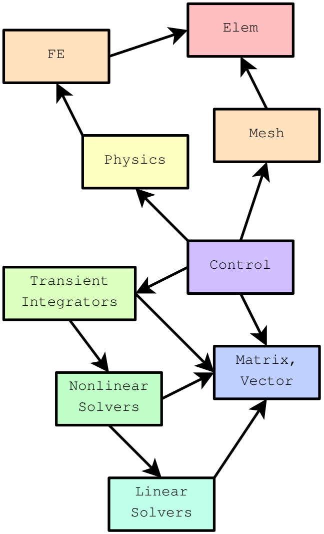


MOOSE – Multiphysics Object Oriented Simulation Environment

- A framework for solving computational nuclear engineering problems in a well planned, managed, and coordinated way
  - *Leveraged across multiple programs*
- Designed to significantly reduce the expense and time required to develop new applications
- Designed to develop analysis tools
  - *Uses very robust solution methods*
  - *Designed to be easily extended and maintained*
  - *Efficient on both a few and many processors*
- Currently supports ~7 applications which are developed and used by ~20 scientists.



Modular Programming



Discrete Components, Interfaces

- Linear, nonlinear solvers are discretization-independent
- System assembly, solution I/O & postprocessing can be discretization-independent
- Time, space discretizations can be physics-independent
- Some error analysis, sensitivity methods can be physics-independent

- Reusable components get re-tested
- Errors too subtle to find in complex physics are easy to spot in benchmark problems.

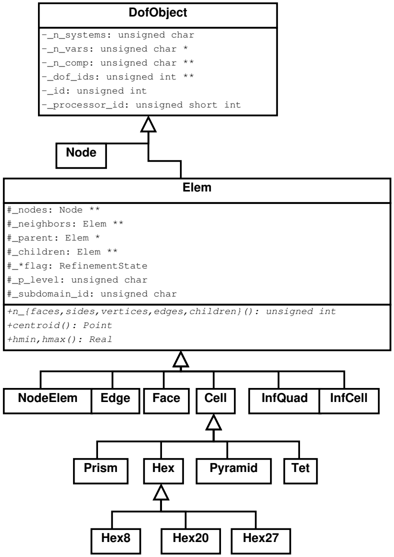
Object Oriented Programming

ABC: Abstract Base Class

- One abstract interface
- Many instantiations
- Hides derived type from most uses

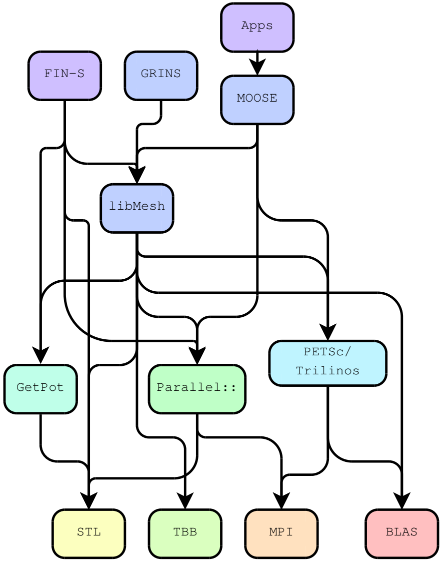
Example: Geometric elements

- Base classes give DoF indexing, mesh topology
- Instantiations give mesh geometry
- Most Mesh code is element-independent



Software Reuse

- Don't reinvent the wheel unnecessarily!
- Time spent rewriting something old is time that could have been spent writing something new.
- More eyes == fewer bugs
- Extensions interoperate



Collaboration Styles

How does collaborative `libMesh` discussion and development take place?

Collaboration Styles

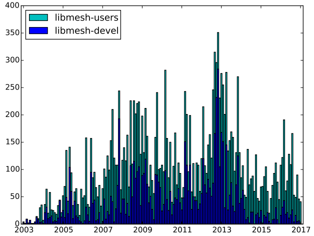
How does collaborative `libMesh` discussion and development take place?

- Yelling at the guy on the other side of the CFDLab

Collaboration Styles

How does collaborative `libMesh` discussion and development take place?

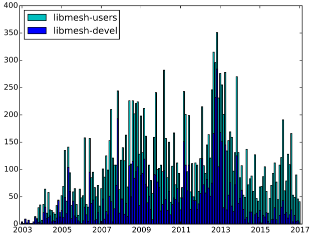
- Yelling at the guy on the other side of the CFDLab
- User, developer mailing lists

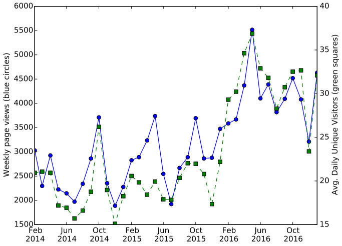


Collaboration Styles

How does collaborative `libMesh` discussion and development take place?

- Yelling at the guy on the other side of the CFDLab
- User, developer mailing lists
- libMesh, MOOSE, GRINS issue trackers

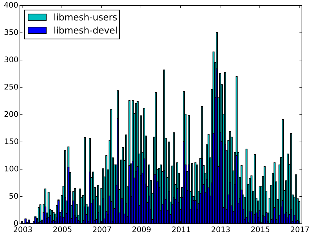


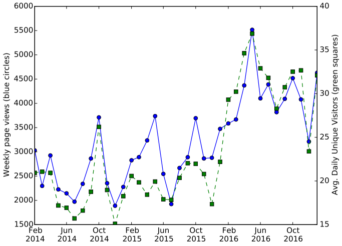


Collaboration Styles

How does collaborative `libMesh` discussion and development take place?

- Yelling at the guy on the other side of the CFDLab
- User, developer mailing lists
- libMesh, MOOSE, GRINS issue trackers
- Private email? Instant messaging? Videoconferencing?





Tracking API Changes

API versions easily proliferate...

```
#if PETSC_VERSION_LESS_THAN(3,1,0)
  ierr = MatGetSubMatrix(matrix->mat(),
           _restrict_to_is, _restrict_to_is_complement,
           PETSC_DECIDE, MAT_INITIAL_MATRIX, &submat1);
#else
  ierr = MatGetSubMatrix(matrix->mat(),
           _restrict_to_is, _restrict_to_is_complement,
           MAT_INITIAL_MATRIX, &submat1);
#endif
```

- Maintain a wide range of external compatibility
  - Dropped PETSc 2.3.3 (2007) support for libMesh 1.0 (2016)
  - C++11 shims
- Limit `libMesh` API changes

Signaling API Changes

Development practices

- Old, new APIs *overlap*
- Easier with C++ function overloading, default arguments
  - Adding `f(a,b)` does not preclude keeping `f(a)`
  - Adding `f(a,b=default)` can replace `f(a)`

Runtime warnings

- `libmesh_experimental()` (in-flux APIs)
- `libmesh_deprecated()` ($\sim$1 year, 1-2 releases)

Examples

- `OStringStream` workaround class
- `Parallel::` global functions

Verification

Does everyone's interpretation of API *semantics* match?

Verification

Does everyone’s interpretation of API *semantics* match?

“When in trouble when in doubt, run in circles scream and shout.”
- old Army War College football team slogan

Verification

Does everyone's interpretation of API *semantics* match?

"When in trouble when in doubt, run in circles scream and shout."
- old Army War College football team slogan

```
#if IN_DOUBT {
  if (in_trouble()) {
    run_in_circles(); // Stack traces, data printing
    scream_and_shout(); // Exception throw
  }
#endif
```

Verification

Does everyone's interpretation of API *semantics* match?

"When in trouble when in doubt, run in circles scream and shout."
- old Army War College football team slogan

```
#if IN_DOUBT {
  if (in_trouble()) {
    run_in_circles(); // Stack traces, data printing
    scream_and_shout(); // Exception throw
  }
#endif

assert(!in_trouble());
```

Verification

Does everyone's interpretation of API *semantics* match?

"When in trouble when in doubt, run in circles scream and shout."
- old Army War College football team slogan

```
#if IN_DOUBT {
  if (in_trouble()) {
    run_in_circles(); // Stack traces, data printing
    scream_and_shout(); // Exception throw
  }
#endif
```

```
assert(!in_trouble());
```

- Each new assertion becomes a new "contract"

High-level Assertions

`libmesh_assert()`, `PETSc` debug mode

- Function preconditions - are arguments all valid?
- Function postconditions - is result valid?
- Active in “debug” or “devel” runs
- Approx. 7000 asserts in `libMesh`; more in GRINS, MOOSE

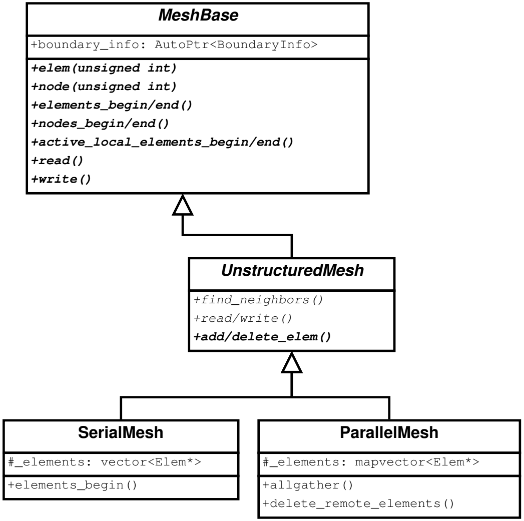
High-level Assertions

libmesh_assert(), PETSc debug mode

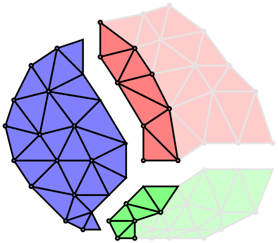
- Function preconditions - are arguments all valid?
- Function postconditions - is result valid?
- Active in “debug” or “devel” runs
- Approx. 7000 asserts in libMesh; more in GRINS, MOOSE

```
libmesh_assert(neigh->has_children());
libmesh_assert(this->initialized());
libmesh_assert((ig >= Ug.first_local_index()) &&
               (ig < Ug.last_local_index()));
libmesh_assert(requested_ids[p].size() == ghost_objects_from_proc[p]);
libmesh_parallel_only(mesh.comm());
MeshTools::libmesh_assert_valid_node_procids(mesh);
libmesh_assert(error_estimator.error_norm.type(var) == H1_SEMINORM ||
               error_estimator.error_norm.type(var) == W1_INF_SEMINORM)
libmesh_assert(number_h_refinements > 0 || number_p_refinements > 0);
```

Upgraded DistributedMesh Support



- `MeshBase` gives node or element iterators
- `ReplicatedMesh` or `DistributedMesh` manages synchronized or distributed data
- Redistribution, AMR/C, etc handled via library



Physics via C++14 Generic Programming

- **C++98:** intrusive metaprogramming

```
template <typename T1, typename T2, typename T3>
typename PlusType<typename MultipliesType<T1,T2>::type,
                  typename ExpType<T3>::type>::type
f(const T1& m, const T2& x, const T3& b)
{ return m*x+exp(b); }
```

- **C++14:** user-friendly return type deduction

```
template <typename T1, typename T2, typename T3>
auto f(const T1& m, const T2& x, const T3& b)
{ return m*x+exp(b); }
```

Physics via C++14 Generic Programming

- Expression-template-compatible kernels:

```
template <typename ContextType,
         typename CacheType>
auto weak_interior_residual
  (const ContextType& context,
   const CacheType&) const
  {
    auto& du_dx  = std::get<1>(context.u);
    auto& v_vals = std::get<0>(context.v);

    return _b*du_dx*v_vals;
  }
```

- `Eigen::Array` calculations auto-vectorize
- `vex::vector` calculations run on GPU
- `MetaPhysicL::DualExpression` calculations provide Jacobian too

Geometric Multigrid Support

- Leverage PETScDM interface for solver infrastructure
- libMesh provides mesh hierarchies, prolongation and restriction operators between meshes
- Future API for user-specified projection operators
- Applicable to any libMesh-based application compiled with PETSc

Number of Levels	2	3	4	5	6	7	8
1-D Laplace	4	4	8(,2)	9(,2)	9(,2)	9(,2)	9(,2)
2-D Laplace (Quads)	5	5	5	5	5	5	5
3-D Laplace (Hexes)	4	5	6	6	5	5	-
2-D Laplace (Tris)	5	7	7	7	7	7	7
3-D Laplace (Tets)	6	8	9	9	-	-	-

(,2) indicates a second outer solver iteration

Acknowledgements

Recent `libMesh` contributors:

- David Andrs
- Paul Bauman
- Vikram Garg
- Derek Gaston
- Dmitry Karpeev
- Benjamin Kirk
- David Knezevic
- Cody Permann
- John Peterson
- Sylvain Vallaghe

Useful resources:

- libMesh: `https://libmesh.github.io/`
- MOOSE: `https://mooseframework.org/`
- GRINS: `https://grinsfem.github.io/`

Dr. Graham F. Carey: ”No one ever got a Ph.D. from here for writing a code.”