Cardinal: A framework for doing FSI with deal.II + IBAMR

David Wells

University of North Carolina, Chapel Hill

In collaboration with:
Laryssa Abdala, Marshall Davey, Simone Rossi, Boyce Griffith

August 12 2024
deal.II Workshop

Goals of this talk

- Talk about what worked and didn't work in writing a big new deal.II app (40k SLOC)
- Focused on new deal.II users (everyone here should learn something)
- Please ask questions, argue with me, etc: I've succeeded if I run out of time, not slides

Governing Equations

Uses the immersed-boundary finite element / finite difference method (IFED) (Boffi and Heltai)

- *Eulerian* finite difference method for Navier-Stokes
- *Lagrangian* finite element method for mechanics

$$\rho \frac{D\boldsymbol{u}}{Dt}(\boldsymbol{x},t) = -\nabla p(\boldsymbol{x},t) + \mu \nabla^2 \boldsymbol{u}(\boldsymbol{x},t) + \boldsymbol{f}(\boldsymbol{x},t),$$

$$\nabla \cdot \boldsymbol{u}(\boldsymbol{x},t) = 0$$

$$\boldsymbol{f}(\boldsymbol{x},t) = \int_{\Omega_0^s} \boldsymbol{F}(\boldsymbol{X},t)\, \delta(\boldsymbol{x} - \chi(\boldsymbol{X},t))\, \mathrm{d}\boldsymbol{X}$$

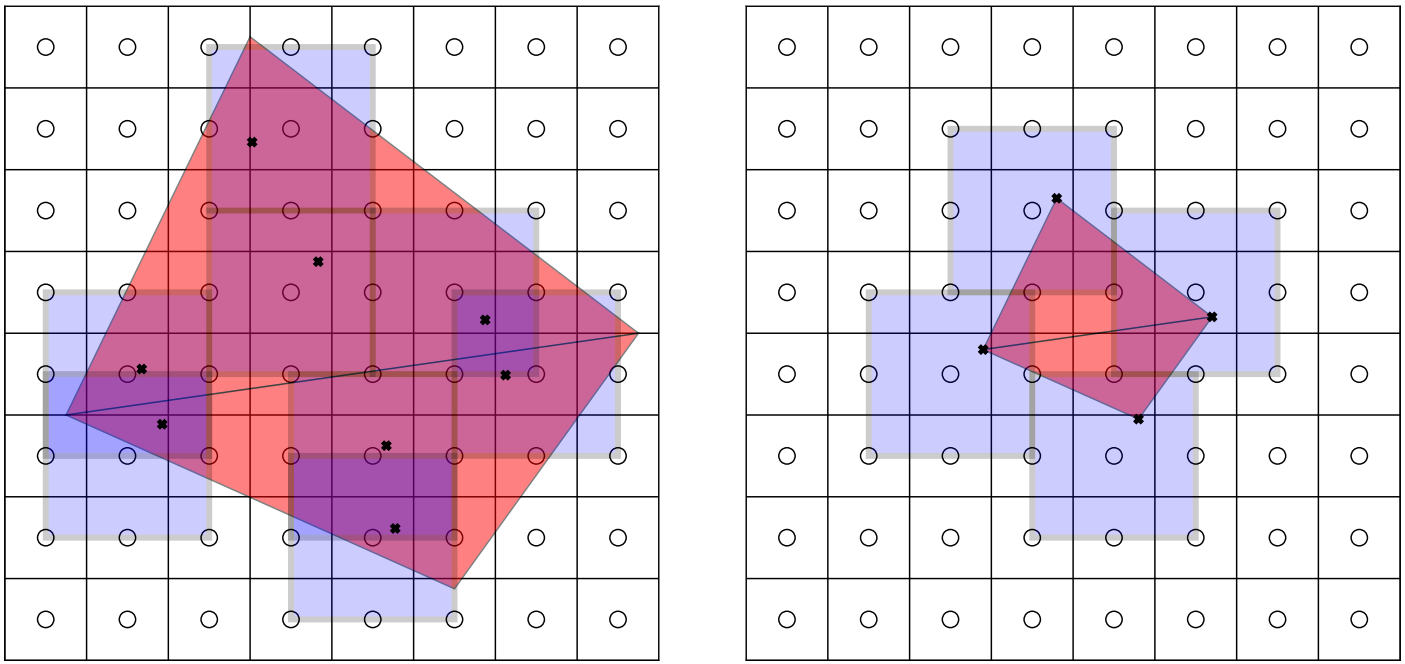
$$\frac{\partial \chi}{\partial t}(\boldsymbol{X},t) = \boldsymbol{U}(\boldsymbol{X},t) = \int_{\Omega} \boldsymbol{u}(\boldsymbol{x},t)\, \delta(\boldsymbol{x} - \chi(\boldsymbol{X},t))\, \mathrm{d}\boldsymbol{x}$$

$$\int_{\Omega_0^s} \boldsymbol{F}(\boldsymbol{X},t) \cdot \psi(\boldsymbol{X})\, \mathrm{d}\boldsymbol{X} = -\int_{\Omega_0^s} \mathbb{P}^e(\boldsymbol{X},t) : \nabla_{\boldsymbol{X}} \psi(\boldsymbol{X})\, \mathrm{d}\boldsymbol{X}$$

The structure's velocity $\frac{\partial \chi}{\partial t}$ is *interpolated* from the fluid $\boldsymbol{u}$; the force on the fluid $\boldsymbol{f}$ is *spread* from the structure $\boldsymbol{F}$.

Regularized Delta Functions

Heart of the IB method: we replace $\delta(\boldsymbol{x})$ with $\delta_h(\boldsymbol{x})$. Here, $\delta_h(\boldsymbol{x})$ has a width of $2\Delta x$.



Can either evaluate the integral operators with Gauss quadrature (left) or nodal quadrature (right). Not shown to scale!

IBAMR implements FDM and IB methods - use deal.II for all unstructured grid things.

Explicit Mechanics

The implicit IB method is WIP for us. We do everything explicitly.

- Disadvantage of IB method: keeping structures in-place is hard
- Body forces (volumetric penalties, spring forces) (`cardinal::ForceContribution`)
- Stresses (first Piola-Kirchoff: $\mathbb{P}^{\mathrm{e}}(\boldsymbol{X}, t)$ from the equations slide) (also `cardinal::ForceContribution`)
- Active strain models (don't use $\mathbb{F}(\boldsymbol{X}, t)$ directly: 'fix' it to only permit deformations in specified directions) (`cardinal::ActiveStrain`)

`dealii::FEValues` is nice, we have `cardinal::MechanicsValues` to compute invariants, strains, etc. (and prevent users from calling `slowpow`)

Goals

What is this project all about? Haven't people already done everything with the IB method?

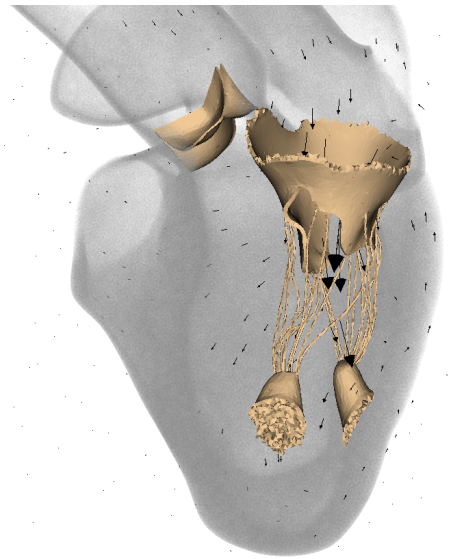
- Immersed methods (not cut cells), neutral buoyancy. IFED now, IIM soon.
- Modern fiber-reinforced material models (e.g., Holzapfel-Gasser-Ogden).
- All features are available through input files.
- Parallel (FEBio has more features but will probably never support MPI).
- Students can do interesting things with it and *not* write 5000 line monoliths

Brief History of Cardinal

How did we get here?

- 2018: four-chambered-heart. Realistic valve dynamics with libMesh!
- 2020: deal.II with simplices is “not impossible” (what did we write on the wiki?)
- 2021: four-chambered-heart, IBAMR, deal.II (fiddle)
- 2022: I write another libMesh-based paper on nodal interaction
- 2023: Heart model (Davey) plus IFED (me) plus electrocardiology (Abdala, Rossi)
- 2024: Realistic valve dynamics with deal.II

What is this talk about?



- Problems Abdala and I ran into
- Interesting Cardinal features
- Ideas for improving other deal.II apps

Framing: eliminating the three kinds of waste (muri, muda, mura).

Who are we?

We are...

- Faculty members, research scientists, postdocs
- Decades of C++ experience
- Formal training in mathematics, computer science

Our jobs are...

- dev ops (who is going to fix the CI?)
- write proposals
- teach classes

We can do anything, but we can't do everything.

Who are graduate students?

Put a good person in a bad system and the bad system wins, no contest.
W. Edwards Deming

Students are...

- Smart
- Know a lot about subject areas (biomechanics, electrophysiology)
- New to scientific computing

Students do not have...

- Years of C++ experience
- Years of engineering experience
- Formal training in computer science

Muri - Unreasonable, Over-burdening

Rule-of-thumb: students should succeed 85% of the time, fail 15% of the time
Which tasks do students struggle with the most?

- C++
- Architecture (we should be the vanguard and rearguard)
- Installing things (students shouldn't spend days recompiling stuff)

Muri - Unreasonable, Over-burdening

C++ is hard. How can we make it easier for students?

- `clang-format` (thanks Matthias!)
- Great documentation (deal.II's greatest strength)
- `-Wall -Wpedantic -Werror -DDEBUG` on the CI
- Constant-time accessors are named `get_*()` (guaranteed to not use MPI, can be called in hot loops, etc)
- Most functions are named `compute_*()`
- Don't worry about ADL et al: prefix member variables with `m_`
- No Greek or shorthand, e.g., `bndry`
- I fight hard to keep CGS units everywhere

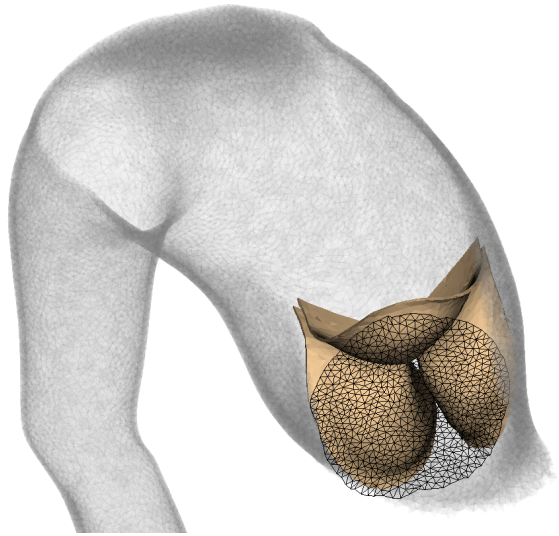
Abdala and I wrote the style guide together - we should keep students involved in writing standards!

Muri - Unreasonable, Over-burdening

C++ is hard. How can we make it easier for students?

- “don’t learn the tricks of the trade, learn the trade”
- ChatGPT frequently solves problems that don’t need to be solved
- TODO: compile more good resources (keynotes, lecture series, books) for students to learn from instead of whatever ChatGPT recommends

Muri - Unreasonable, Over-burdening



Provide good architecture so students don't have to solve every problem, just their problem.

- I spent a lot of time reading Aspect's source code (thanks!)
- Make it possible for students to get the job done by inheriting from a class, using a plugin, etc. instead of patching some huge class

Overburdening: Use Good Architecture

How do we prevent students from writing 5000-line monoliths? How do we sustainably grow deal.II? What is deal.II?
deal.II architecture:

- The very nice dependency graph on the front page of the manual
- `dealii::SmartPointer` and `dealii::Subscriptor`: one-to-many
- No global variables!
- All time-dependent state is in vectors, which are loosely coupled via `dealii::Utilities::MPI::Partitioner`

Overburdening: Use Good Architecture

Cardinal architecture:

- 'When in Rome': inherit several IBAMR-isms (SAMRAI's input files, restart singleton class)
- RAII, no staged initialization (not obvious to students!)
- Who is responsible? `std::move()` when we can, `subscribe()` if we must, `std::shared_ptr` if absolutely necessary
- Like deal.II: most of the work happens in utility functions, classes (except FE vectors) are immutable modulo regridding
- Goal: everything accessible through the input file. Custom code via plugins

Overburdening: Use Good Architecture

Cardinal architecture:

- Like Aspect: `main.cc` sets up the `cardinal::Simulator` God-object (unavoidable to write a reasonable time-stepping loop), catches exceptions
- Unlike Aspect: no `dlopen()` for plugins - they are explicitly passed to `cardinal::Simulator` (not sure which is better)
- Unlike Aspect: no `cardinal::SimulatorAccess` yet

Overburdening: Use Good Architecture

Cardinal architecture: `cardinal::Part<dim, spacedim>`

- solely responsible for the position, velocity, and material models (forces, active strains) of one structure
- ctor:
  `std::vector<std::unique_ptr<cardinal::ForceContribution<dim>>>`,
  `std::vector<std::unique_ptr<cardinal::ActiveStrain<dim>>>`,
- uses RAII
- only place where we use `set_*()` functions!
- `SmartPointer<const cardinal::Part<dim>>` grants access

Overburdening: Use Good Architecture

Cardinal architecture: `cardinal::fsi::IFEDMethod<dim>`

- inherits from `IBAMR::IBStrategy`
- 'handed-off' to `IBAMR::IBExplicitHierarchyIntegrator`
- ctor: `std::vector<cardinal::Part<dim>>` (unique ownership of `cardinal::Part`s)
- implementation: about 1500 LOC, all loops over cells are in utility functions

Overburdening: simplify compilation

Students spend a *shocking* amount of time recompiling stuff! Everything that can go wrong will frequently go wrong.

- Multiple copies of PETSc
- Conflicts between system libraries and user libraries
- Don't know what package managers are

Solution: autoibamr (forked from candi). Minimal configuration options (known-good versions, only three build types) means it is more likely to succeed. We should continue investing in anything that makes this process easier.

Muda - non-value-added work

The most dangerous kind of waste is the waste we do not recognize.

Sigeo Shingo

What do we actually do that is of value?

- Graduate students
- Write papers
- Ship software

Everything else is either necessary (e.g., write grants) or unnecessary (should be removed).

non-value-added work: observations

What are our students actually doing?

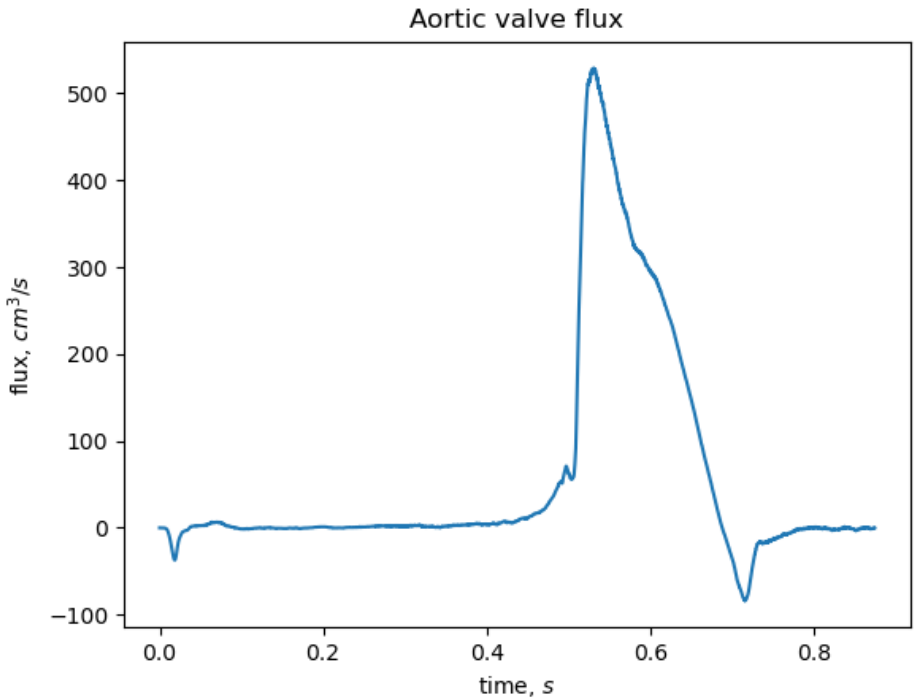
- Compiling code (*fixes*: ccache, CI, better hardware, use `-O1` or `-Os`)
- Debugging (*don't write bugs!*)
- Running everything in debug mode (too slow!)
- Managing dependencies, upgrades, system administration
- Typing too slowly or inaccurately

non-value-added work: compiling

Various performance improvements are not obvious - students don't have the vocabulary.

- `make` to `ninja` (what is `-j4`?)
- `-O3` to `-O1` (multiple builds?)
- `ccache` (what are object files?)
- Interesting linker developments (`ld`, `gold`, `mold`)
- Make the CI fail fast
- Trickery to make `-DDEBUG` work with `-O1` in release mode

non-value-added work: munging text files



four-chambered-heart: Need to log something? Open a text file!

- Produces wrong output with restarts!
- What went where and why? How do we pick file names?
- Are we using mmHg today? Are you *sure* it is mmHg?

non-value-added work: munging text files

We have a single class now for all output: `cardinal::StatisticalOutput`

```
template <int dim>
struct Statistics
{
  std::string                                                group;
  std::map<std::string, std::pair<double, Unit>>             scalars;
  std::map<std::string, std::pair<Tensor<1, dim>, Unit>> vectors;
};
```

- Quite different from the Aspect version - uses 100% HDF5 (structured output, stores units as metadata, etc), no `TableHandler`
- Works correctly with restart files
- IBAMR-ism: can load the HDF5 file as an input file
- Future work: callbacks, user-defined statistics via plugins

Mura - patches, unevenness, non-uniformity

Where there is no Standard there can be no Kaizen. Taiichi Ohno

- Georgetown's Camry plant: all jobs equally physically taxing, rotate to avoid RSI
- Bowling Green's Corvette plant: younger staff get harder jobs, more senior staff pick less taxing jobs

Unevenness: definitions in scientific computing

What does this mean in scientific software?

- Try to level out the workload (e.g., having the CI automatically file issues)
- Minimize time between error creation and error detection
- Minimize time to merge new code
- Train students to break up work
- I refuse to review patches with more than 1k lines of source code changed

Unevenness: case example

```
for (unsigned int i = 0; m_ionic_parts.size(); i++)
    m_ionic_parts[i].save(archive, version);
```

What went wrong here? Let's do 5-why!

```
An error occurred in line <1050> of file <[...]/source/base/utilities.cc>
 in function
 void [...]::posix_memalign([...])
The violated condition was:
 ierr == 0
Additional information:
 Your program tried to allocate some memory but this allocation failed.
 Typically, this either means that you simply do not have enough memory
 in your system, or that you are (erroneously) trying to allocate a
 chunk of memory that is simply beyond all reasonable size, for example
 because the size of the object has been computed incorrectly.

 In the current case, the request was for 18410715276690587648 bytes.
```

Unevenness: case example

My program crashed in debug mode.

1. Why? My vector tries to allocate 16 exabytes of memory.
2. Why? Clearly (to an expert) reading uninitialized memory.
3. Why? Walked off the end of an array.
4. Why? No range checks.
5. Why? We don't have a way to turn on range checks automatically

Now what?

- Starter fix: compile everything with `-D_GLIBCXX_ASSERTIONS` (easy)
- Better fix: set up CI to use `-D_GLIBCXX_DEBUG_PEDANTIC`, sanitizers (harder)