PETSc

PETSc is a set of libraries for the efficient, scalable, solution of systems of nonlinear algebraic equations, and an extensible platform for scientific computing.

- **Modular**: Functionality is cleanly separated into interacting interfaces
- **Scalable**: Code runs efficiently on 1–1,000,000+ cores
- **Extensible**: Users can easily add functionality to solve their problems

Clean Hierarchy of Interfaces

For each PETSc class, there is a toplevel interface, using only other toplevel interfaces, and internally we only use these interfaces when employing other classes.

PETSc dispatches to an implementation class from the toplevel interface, allowing us to switch storage formats on the fly, or in the case of solvers to change algorithms. We can seamlessly manage external hardware and storage.

Why Use Dynamic Dispatch?

- Runtime customization of implementation
- Smaller, abstract interface
- No runtime type checks for customization
- Single breakpoint for debugging

Why Not Use Templates?

- Lack of encapsulation (above and below template library interfaces)
- Crowds namespace and interfaces with instantiated names
- Template expansion code not available to the user
- Compile time and error message explosion
- Convoluted instantiation and resolution logic

Why Dynamic Library?

- Runtime customization of implementation
- Smaller, abstract interface
- No runtime type checks for customization
- Single breakpoint for debugging

Why Use Plugins?

- Inversion of Dependencies
  - The base `library` implementation has a non-circular dependency, so it cannot be in a static library, but it does belong in the solver library. We can put it in a third library that uses the solver library dynamically.

- Extensibility without recompilation or reinstallation
- Static compilation does not provide a performance boost

PETSc objects can be configured at runtime with Options Database, the Service Locator pattern. For example, an optimized solver for the 2D Allen-Cahn problem can be constructed entirely from options:

```
library: gmres
ksp_type: chebyshev
chebyshev_degree: 5
preconditioner: mg
mg_levels: 1
mg_coarse_ksp_type: preonly
mg_coarse_pc_type: svd
```

PETSc objects provide modular and efficient solvers for a wide range of applications, from fluid dynamics to structural mechanics. They are designed to be portable, efficient, and scalable, making them a popular choice in computational science.

Configure Extensibility

PETSc Configure can be extended with user modules at runtime. The tests for many packages are now contributed, and work automatically when dropped in the modules directory. Advantages of our system over others, such as Autotools and SCons:

1. **Namespace**
   - Tests are wrapped up in modules, which also hold the test results. Thus, you get the normal Python namespace of modules. As simple as this sounds, SCons does not do it. For CMake, not even CMake. They use a file namespace. Also, when we build up timestamped, you can see where systems come from, whereas in the others, all files are dumped into random files like `include` and `lib`.

2. **Explicit control flow**
   - The modules are organized explicitly in a DAG. The user indicates dependencies with a single call, `ccode` or `ccode ...`, which not only structures the DAG, but also renames the output files to reflect the module.

3. **Multi-language tests**
   - We have explicit testing and packaging of languages, so builds can use any code they want, with all their own compile, flags, libraries, etc. Thus, it is easy for us to do cross-language checks in a few lines, whereas this is very difficult in other systems.

4. **Subpackages**
   - We have a template package (and a class specialization) that PETSc downloads, builds, and tests immediately. In some cases, even people use packages through PETSc, because it will get build and tested automatically. Most other systems have no idea of packaging.