Fortran package manager
Toward a rich ecosystem of Fortran packages

Sebastian Ehlert, Ondřej Čertík, Milan Curcic, Jakub Jelínek, Laurence Kedward, Vincent Magnin, Emanuele Pagone, Brad Richardson, John Urban

24th September, 2021
FortranCon
Outline

1. Motivation
2. Implementing fpm in Fortran
3. Feature overview
4. Summary and outlook
Motivation

- building and distributing Fortran software is difficult
  - manual makefiles are seldom cross platform compatible
  - autoconf is not working on native Windows
  - learning curve for CMake is incredibly steep
  - meson does not install module files automatically
- writing of build files and adjustment for Fortran is time-intensive
- difficult and tedious to reuse Fortran source code
- system-wide installation dangerous for multiple Fortran compilers (no ABI compatibility)
- impossible to depend on a project without redistributing it
  ▶ create a Fortran-specific build system and package manager to fix this
  ▶ make it easy to create reusable Fortran libraries
  ▶ Fortran package manager (fpm)
Adoption of fpm

- today **173** open source projects are using fpm
- ca. **160** unique developers have contributed to those projects
- **53** contributors at the fpm project
- code contributions from **19** developers to fpm
- hosted on GitHub (MIT licensed):
  - [https://github.com/fortran-lang/fpm](https://github.com/fortran-lang/fpm)

Data collected from GitHub and GitLab on 15th Sep, 2021
Language choice for implementing a package manager

- fast without much overhead
- self-contained and simple to setup
- easily accessible for users/contributors

- use compiled language
- small runtime library or static binaries
- mostly Fortran developers

- **Fortran** is the most well-known language in our community
- compilers can easily produce **static binaries** for distribution
- we can identify short-comings in Fortran while implementing fpm
- direct feedback for useful features in **stdlib**
Building blocks for Fortran implementation

1. command-line interface builder
   - M_CLI2 package
2. TOML parser library
   - toml-f package
3. Fortran source scanning
   - implemented in fpm
4. multiprocessing and scheduling
   - OpenMP directives
Bootstrapping fpm

- fpm is built with itself
- single-file version used for creating initial fpm bootstrap binary
- minimal dependencies, requires only a Fortran compiler to build
- bootstrap binary is used to build fpm: Fortran, C compiler and git are required
Command-line features

- handles (sub)module interdependencies
- automatically finds executables and tests
- runs executables and examples
- wildcard globbing for selection
- finds and runs test executables
- allows running tests in debugger
- automatically installs project
- user prefix used by default
- creates new projects with fpm layout
- backfilling to update existing projects
- updates all dependencies
- basic caching and locking
Layout of an fpm project

- package manifest in **TOML**
- limited complexity of build file
- mainly meta data of project (license, author, keywords, …)
- default layout requires only **name** (src, app, example, and test)
- **source-dir** for customizing layout
- executable names from **program** unit
- system libraries can be linked
First-class dependencies

- dependencies are currently specified by referencing another project (e.g. via git URL)

```ini
name = "lib"
[dependencies]
toml-f.git = "https://github.com/toml-f/toml-f"
stdlib = {git="https://github.com/fortran-lang/stdlib", branch="stdlib-fpm"}

[dev-dependencies]
test-drive.git = "https://github.com/fortran-lang/test-drive"

[[executable]]
name = "runner"
[executable.dependencies]
M_CLI2.git = "https://github.com/urbanjost/M_CLI2"
```

- different scopes available beside full dependencies
- can depend on projects only for testing or limit dependencies to single executable
Extending fpm with plugins

- package manifest provides space for third-party projects
- fpm is available as fpm package and can be used as dependency
- plugins are used automatically from the command-line
- useful for staging new features before integration in fpm

Searching for fpm packages: fpm-search
- available from fpm search command
- downloads the fpm registry
- search package names, descriptions

Documentation of intrinsics: fpm-man
- available from fpm man command
- access to documentation of intrinsics
- cross-platform, no man required
Near-future roadmap

- compiler profile support
- custom preprocessor step (especially for fypp projects, i.e. stdlib)
- support for a package registry in fpm
- optional dependencies
- build script support for complex packages
Get fpm now

```bash
❯ conda create -n fpm fpm
❯ conda activate fpm
❯ spack install fpm +openmp
❯ spack load fpm +openmp
❯ brew tap awvwgk/fpm
❯ brew install fpm
❯ git clone https://github.com/fortran-lang/fpm
❯ cd fpm && ./install.sh
```