Spack Package Manager

Introduction and Best Practices

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Michael Kuhn
michael.kuhn@informatik.uni-hamburg.de

2020-03-12

Scientific Computing
Department of Informatics
Universität Hamburg
https://wr.informatik.uni-hamburg.de

About Us: Scientific Computing (Wissenschaftliches Rechnen)

- High Performance Computing
- Storage and Parallel I/O
- Data Reduction Techniques

- Middleware Optimization
- Alternative I/O Interfaces
- Cost and Energy Efficiency

We are an Intel Parallel Computing Center for Lustre
(“Enhanced Adaptive Compression in Lustre”)

Motivation

Introduction

Hands-On

Best Practices

Summary

Motivation

- Installing software can be complicated
  - Figure out and install dependencies
  - Get familiar with different build systems
  - Debug build problems on own architecture
  - Handle different versions and configurations
- Spack is a package manager for supercomputers
  - Works on Linux and macOS
  - You can also use it to build software on your laptop

Motivation...

- Let’s try something simple: m4 (with libsigsegv support)

Motivation. . .

- Let’s try something simple: m4 (with libsigsegv support)
  1. Download and extract libsigsegv

     ```
     $ wget ... && tar xf ...
     ```

Motivation…

- Let’s try something simple: m4 (with libsigsegv support)
  1. Download and extract libsigsegv

     ```
     $ wget ... && tar xf ...
     ```

  2. Build and install libsigsegv

     ```
     $ ./configure --prefix=... && make && make install
     ```

Motivation...

- Let’s try something simple: m4 (with libsigsegv support)
  1. Download and extract libsigsegv

     ```
     $ wget ... && tar xf ...
     ```

  2. Build and install libsigsegv

     ```
     $ ./configure --prefix=... && make && make install
     ```

  3. Download and extract m4

     ```
     $ wget ... && tar xf ...
     ```

Motivation...

- Let’s try something simple: m4 (with libsigsegv support)
  1. Download and extract libsigsegv

     ```
     $ wget ... && tar xf ...
     ```
  2. Build and install libsigsegv

     ```
     $ ./configure --prefix=... && make && make install
     ```
  3. Download and extract m4

     ```
     $ wget ... && tar xf ...
     ```
  4. Figure out how to specify dependency

     ```
     $ ./configure --help
     ```

Motivation...

- Let’s try something simple: m4 (with libsigsegv support)

  1. Download and extract libsigsegv

     ```
     $ wget ... && tar xf ...
     ```

  2. Build and install libsigsegv

     ```
     $ ./configure --prefix=... && make && make install
     ```

  3. Download and extract m4

     ```
     $ wget ... && tar xf ...
     ```

  4. Figure out how to specify dependency

     ```
     $ ./configure --help
     ```

  5. Build and install m4

     ```
     $ ./configure --prefix=... --with-libsigsegv-prefix=... &&
       ↪ make && make install
     ```

Motivation. . .

- Now let’s do the same with Spack

Motivation…

- Now let’s do the same with Spack
  1. Install m4 and all dependencies

     ```
     $ spack install m4
     ```

Motivation…

- Now let’s do the same with Spack
    1. Install m4 and all dependencies

       ```
       $ spack install m4
       ```

- Real software typically has more dependencies
- Users might want several versions (Python 2.x and 3.x)
- Access to software should be easy (modules)

Motivation

Introduction

Hands-On

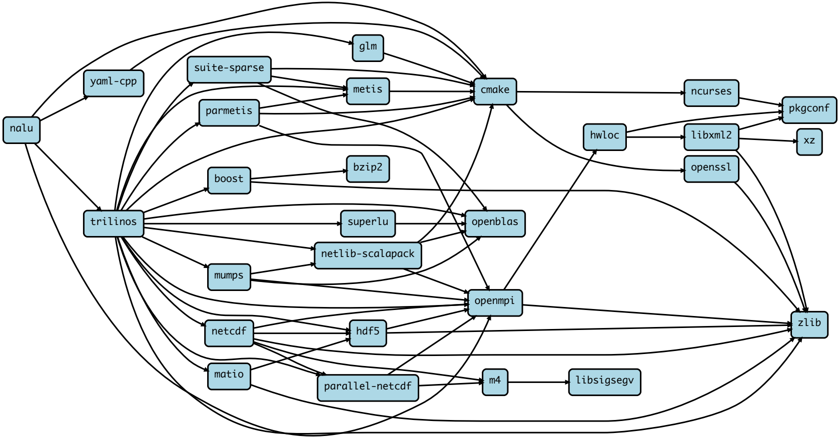
Best Practices

Summary

Introduction

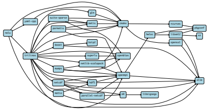
- The following five slides are taken from the Spack tutorial:
  - `https://spack-tutorial.readthedocs.io/`
- For more information, take the tutorial
  - Covers more ground and explains features in more detail
  - Regularly held at SC (USA) and ISC (Frankfurt)

Software complexity in HPC is growing

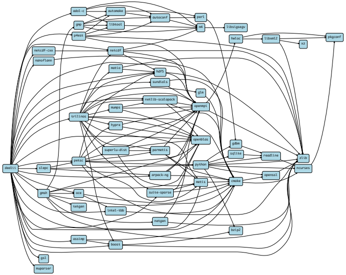


Nalu: Generalized Unstructured Massively Parallel Low Mach Flow

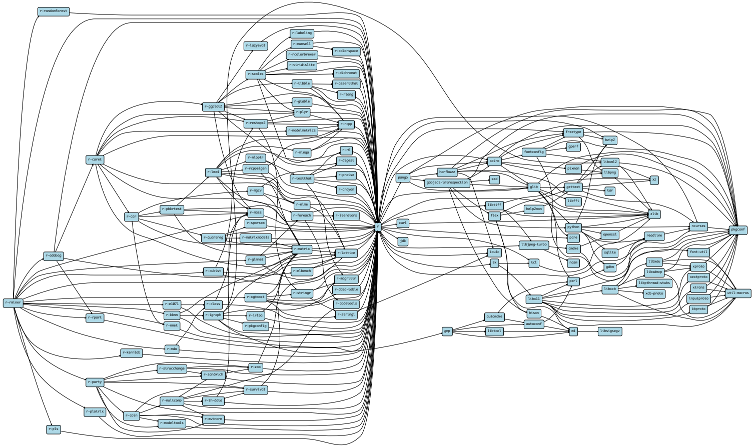
Software complexity in HPC is growing



Nalu: Generalized Unstructured Massively Parallel Low Mach Flow



dealii: C++ Finite Element Library



R Miner: R Data Mining Library

The complexity of the exascale ecosystem threatens productivity.

15+ applications x **80+ software packages** x **5+ target architectures/platforms**
Xeon Power KNL
NVIDIA ARM Laptops?

x **Up to 7 compilers**
Intel GCC Clang XL
PGI Cray NAG

x **10+ Programming Models**
OpenMPI MPICH MVAPICH OpenMP CUDA
OpenACC Dharma Legion RAJA Kokkos

x **2-3 versions of each package**
+ external dependencies

= up to **1,260,000** combinations!

- Every application has its own stack of dependencies.
- Developers, users, and facilities dedicate (many) FTEs to building & porting.
- Often trade reuse and usability for performance.

We must make it easier to rely on others' software!

Who can use Spack?

People who want to use or distribute software for HPC!

1. **End Users of HPC Software**
   - Install and run HPC applications and tools
2. **HPC Application Teams**
   - Manage third-party dependency libraries
3. **Package Developers**
   - People who want to package their own software for distribution
4. **User support teams at HPC Centers**
   - People who deploy software for users at large HPC sites

Spack is being used on many of the top HPC systems

- Official deployment tool for the U.S. Exascale Computing Project
- 7 of the top 10 supercomputers
- High Energy Physics community
  - Fermilab, CERN, collaborators
- Astra (Sandia)
- Fugaku (Japanese National Supercomputer Project)

ECP
EXASCALE COMPUTING PROJECT

富岳

Fugaku coming to RIKEN in 2021
DOE/MEXT collaboration

Summit (ORNL), Sierra (LLNL)

SuperMUC-NG (LRZ, Germany)

Edison, Cori, Perlmutter (NERSC)

Motivation

Introduction

Hands-On

Best Practices

Summary

Installation and Setup

- Clone Spack and use current stable branch

```
$ git clone -b master https://github.com/spack/spack.git
```

Installation and Setup

- Clone Spack and use current stable branch

```
$ git clone -b master https://github.com/spack/spack.git
```

- Enable Spack’s shell integration for easier use

```
$ . spack/share/spack/setup-env.sh
```

Basic Usage

- Get a list of all supported packages

```
$ spack list
```

Basic Usage

- Get a list of all supported packages

```
$ spack list
```

- List only Python modules

```
$ spack list py-
```

Basic Usage

- Get a list of all supported packages

```
$ spack list
```

- List only Python modules

```
$ spack list py-
```

- Show information about a package

```
$ spack info m4
```

Package Installation

- Install package for m4

```
$ spack install m4
```

Package Installation

- Install package for m4

```
$ spack install m4
```

- List all installed packages

```
$ spack find
$ spack find -d
```

Package Installation

- Install package for m4

```
$ spack install m4
```

- List all installed packages

```
$ spack find
$ spack find -d
```

- Load the package to make it usable

```
$ spack load m4
$ which m4
```

Specs and Variants

- Show spec for m4

```
$ spack spec -I m4
```

Specs and Variants

- Show spec for m4

```
$ spack spec -I m4
```

- Show spec for m4 without sigsegv variant

```
$ spack spec -I m4~sigsegv
```

Specs and Variants

- Show spec for m4

```
$ spack spec -I m4
```

- Show spec for m4 without sigsegv variant

```
$ spack spec -I m4~sigsegv
```

- Install m4 without sigsegv support

```
$ spack install m4~sigsegv
```

Specs and Variants

- Show spec for m4

```
$ spack spec -I m4
```

- Show spec for m4 without sigsegv variant

```
$ spack spec -I m4~sigsegv
```

- Install m4 without sigsegv support

```
$ spack install m4~sigsegv
```

- List all installed packages

```
$ spack find
$ spack find -d -v
```

Versions

- Show available versions for libsigsegv

```
$ spack info libsigsegv
```

Versions

- Show available versions for libsigsegv

```
$ spack info libsigsegv
```

- Install m4 with older libsigsegv version

```
$ spack install m4 ^libsigsegv@2.11
```

Versions

- Show available versions for libsigsegv

```
$ spack info libsigsegv
```

- Install m4 with older libsigsegv version

```
$ spack install m4 ^libsigsegv@2.11
```

- List all installed packages

```
$ spack find
$ spack find -d -l -v
```

Modules

- Spack creates modules for each installed package

```
$ spack module tcl find m4 ^libsigsegv@2.11
```

Modules

- Spack creates modules for each installed package

```
$ spack module tcl find m4 ^libsigsegv@2.11
```

- Show available modules

```
$ module avail
```

Modules

- Spack creates modules for each installed package

```
$ spack module tcl find m4 ^libsigsegv@2.11
```

- Show available modules

```
$ module avail
```

- Load module

```
$ module load m4-1.4.18-gcc-9.2.1-4ox26wb
```

Additional Features

- Spack has a wide range of features
  - Multiple compilers
    - `%gcc`, `%clang` etc.
  - Virtual providers
    - `mpi` can be provided by `openmpi`, `mpich` etc.
  - Uninstall packages cleanly
    - Unneeded packages can be removed with the `gc` command
  - Support for containers
    - `containerize` can prepare Dockerfiles
  - Environments and views
    - Bundle together related software

Motivation

Introduction

Hands-On

Best Practices

Summary

Use Cases

- Reminder: Spack can serve different use cases
  1. End users
  2. Application developers
  3. Package developers
  4. Operators
- We will focus mostly on the last use case
  - WR’s software stack is provided by Spack

Considerations

- Be careful when using `develop` in production
    - Each installed package has a hash
        - All (except pure build) dependencies are included in hash
        - Hash calculation is mostly stable nowadays
    - Changes in common dependencies can require rebuilding everything
- Consider overhead on clusters
    - Spack is written in Python and contains a lot of small files
        - Can be slow on network file systems
    - Modules can be used to mitigate performance problems

Management Approaches

1. Keep old software versions
    - Install latest stable release and all required packages
    - Pull in new versions of major packages whenever needed
        - Problematic: security updates, updates of dependencies etc.
2. Reinstall software frequently
    - Frequently install latest stable release and all required packages
    - Keep around old software stacks, announce migration periods
        - Developer time is expensive, CPU and HDD are cheap

Management Approaches...

- Example: WR’s cluster
  - Full software stack is rebuilt regularly
    - Newest stack is labeled as `current`
    - Old stacks are kept for one year and then deleted
  - Modules are named more conveniently (`name/version-hash`)
    - Allows, for example, `module load hdf5`
  - Standard software is loaded on login
    - Current GCC and MPI, can be extended easily
  - Users can use chaining to avoid rebuilding available software
    - Packages from upstream installations can be used seamlessly

Motivation

Introduction

Hands-On

Best Practices

Summary

Summary

- Software management is hard
    - Applications often have dozens of dependencies and configurations
- Spack is a package manager for supercomputers
    - Supports different use cases: users, developers and operators
- Contains packages for a wide range of software
    - Configuration, build and installation are handled by Spack