Installing your software packages with Spack

Pawel Pomorski
ppomorsk@sharcnet.ca

September 10, 2025

Introduction

Spack is a package management tool designed to support multiple versions and configurations of software on a wide variety of platforms and environments

Current Alliance software stack uses Easybuild for package management and there are no plans to change that

Spack could augument the existing Easybuild software stack

Users may already be using Spack and need assistance

https://spack.readthedocs.io/en/latest/

https://spack-tutorial.readthedocs.io/en/latest/

Complexity of software builds

Scientific software getting ever more complex, with many dependencies and variants

In HPC environment should be highly optimized for the hardware it is running on

Needs to be built efficiently and reproducibly, with as little effort as possible

Cluster software stack

A multitude of software packages in different versions needs to be installed in an efficient way

This must be highly automatized for building a huge software stack

Single user installs

Users may need to compile complex software themselves as cluster software stack cannot accomodate everyone, or installation may take time

Linux distribution might not have exactly the versions needed, usually installs single version

Users may need latest versions or particular version combinations to ensure reproducibility for example

Containers are an alternative but those also need to be built

Premade containers might not be optimized for hardware

An automatic build system able to build packages and their dependencies efficienly could be very useful

Spack features

Package manager built for HPC environments

Root access not required

Every unique package installed into its own prefix, multiple packages can easily coexist

Installs from source or from fast from binary build caches

Simple installations: tell Spack briefly what is required and it should figure out the details

Not an operating system, runs on top of existing OS (Linux, macOS, Windows)

Relies on host system for essential libraries

Performs native builds, not cross compilation. Builds software for the same processor architecture it is running on. Support for cross compilation might be added in the future

How could Spack it fit into current Alliance setup?

Easybuild remains the main tool as it works well in our setup

For now the Alliance could explore using Spack to augument software stack for specific applications

Large scale runs on Trillium

Scinet Spack experimental configuration (Mike Nolta)

https://github.com/SciNetHPC/spack-config

Software for novel architectures: AMD APUs on Nibi requiring ROCm builds

Some support for users that want to use Spack in their space

Spack for users

The default Alliance software stack is provided via modules pointing to software built with Easybuild. Users can request software to be installed as a module

However, that process can be time consuming as staff time is limited

In theory users can write build their own Easybuild packages but the process is cumbersome

Spack does not require root access, so users on Alliance clusters can use it to install software

Spack can be a possible alternative if for example many different versions of software need to be built and tested, if very specific versions are required, or if Spack is the preferred way to install some software

Start with Spack

No complicated installation required, just clone and go

```
git clone --depth=2 --branch=releases/v1.0 \
https://github.com/spack/spack.git ~/spack

. ~/spack/share/spack/setup-env.sh
spack repo update builtin --tag v2025.07.0
spack compiler find
spack install the_package_you_want
```

Spack version: use 0.23 or 1.0 ?

Recent milestone release 1.0 (July 2025) made major changes

Package recipes in separate repository now

Some changes in Spack configuration layout, so make sure to consult correct documentation

Some package recipes may not have caught up with the changes, so builds may fail for some packages with the new version

In that case building with version 0.23 could work

Spack storage

Spack installs packages into the directory where it was unpacked by git

~/.spack directory stores configuration files

Temporary build files go into /tmp, but can change that by defining $TMP

Spack may use a lot of space on disk especially if installing many complex packages

Spack compilers

Initial compiler must come from outside Spack (bootstrapping issue)

Such a compiler is an external package in Spack

Spack find all available compilers, though configuration might need tweaking

Can then install your own compilers within Spack

External packages

External packages can be used so not everything that Spack uses must be built by Spack. Convenient if a preferable installation already exists

Somewhat reduces reproducibility, so may want to avoid them if that is a concern

Spack packages

About 8000 packages supported

https://packages.spack.io/

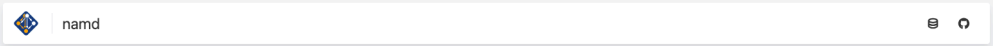
Since version 1.0 build recipe files have a separate repository

https://github.com/spack/spack-packages

Pin your Spack installation to specific release for stability

Possible to write your own recipes to build your software

Example package - NAMD



Description

NAMD is a parallel molecular dynamics code designed for high-performance simulation of large biomolecular systems.

Dependencies

amdfftw, c, charmpp, cuda, cxx, fftw, gmake, hip, hsa-rocr-dev, llvm-amdgpu, mkl, python, tcl

Required by

This package is not required by any other packages

Variants

- amdgpu_target (none) AMD GPU architecture
- avxtiles (false) Enable avxtiles supported with NAMD 2.15+
- build_system (makefile) Build systems supported by the package
- cuda (false) Build with CUDA
- cuda_arch (none) CUDA architecture
- fftw (3) Enable the use of FFTW/FFTW3/MKL FFT/AMDFFTW
- interface (tcl) Enables Tcl and/or python interface
- memopt (false) Enable memory-optimized build supported with NAMD 2.8+
- rocm (false) Enable ROCm support
- single_node_gpu (false) Single node GPU

Patches

inherited-member-2.13.patch when @2.13

inherited-member-2.14.patch when @2.14

HOMEPAGE

Build System

MakefilePackage (view package.py)

Maintainers

- jcphill

Versions

- master
- 2.14
- 2.12
- 3.0.1
- 2.13

Deprecated versions

- 3.0b7
- 3.0b3
- 2.15a2
- 2.15a1
- 3.0b6
- 3.0
- 2.15a1.manual

Conflicts

%clang@7: when +cuda ^cuda@10.0.130~allow-unsupported-compilers target=ppc64le:

%clang@:3.7,6.1: when +cuda ^cuda@10.0.130~allow-unsupported-compilers target=x86_64:

%clang@:3.7,8.1: when +cuda ^cuda@10.1.105:10.1.243~allow-unsupported-compilers target=x86_64:

%clang@:3.7,7.1: when +cuda ^cuda@10.1.105~allow-unsupported-compilers target=x86_64:

%clang@:3.2,9: when +cuda

Package build examples

Wide variety of options available when building packages

```
spack install namd
spack install namd@3.0.1
spack install namd@3.0.1+cuda cuda_arch=90
spack install namd@3.0.1+rocm amdgpu_target=gfx942
```

Sigils

```
+ (enable) and ~ or - (disable)
@ versions
% direct dependencies
^ transitive dependencies (dependency of dependency)

spack install namd@3.0.1 %gcc@14.3.0
spack install namd@3.0.1 cflags=-O3
```

Tracking multiple versions with Spack hashes

```
spack install namd@3.0.1
spack install namd@3.0.1 cflags=-O3

$ spack find -l namd
-- linux-ubuntu24.04-zen4 / %c,cxx=gcc@13.3.0 ---
bbvzgu3 namd@3.0.1  tfrazak namd@3.0.1
==> 2 installed packages

$ spack diff /bbvzgu3 /tfrazak
--- namd@3.0.1/bbvzgu33c4sp2lqq5av63trm2lxibig2
+++ namd@3.0.1/tfrazakna3acyssgjtz36uljj7guq6wv
@@ hash @@
-  namd bbvzgu33c4sp2lqq5av63trm2lxibig2
+  namd tfrazakna3acyssgjtz36uljj7guq6wv
@@ node_flag @@
+  namd node_flag("cflags","-O3","-O3","none")
```

Spack difficulties

Spack may not “just work”

Compare with a Linux distribution that installs only selected versions against a known setup, where everything is tested

Spack configuration space much larger, there are many more permutations of options

Underlying system setup different in each case, may be non-standard on clusters

Some fiddling with install options is usually required except for simple packages

Spack packages maintained by community, so reliability may vary

Spack environments

Somewhat analogous to Python environments, will create a selection of packages to be used for particular task

Executables of packages in the environment will be put in the PATH if possible

```
spack env create myproject
spack env list
spack env activate myproject
spack add tcl
spack add trilinos
spack install
```

Generates spack.yaml (configuration) and spack.lock (provenance)

May need to edit spack.yaml before install step

Start with Spack on Alliance clusters

```
git clone --depth=2 --branch=releases/v1.0 \
https://github.com/spack/spack.git \
/project/def-ppomorsk/ppomorsk/spack

. /project/def-ppomorsk/ppomorsk/\
spack/share/spack/setup-env.sh

export TMP=/scratch/ppomorsk/spacktmp
spack repo update builtin --tag v2025.07.0
spack compiler find
# edit compiler settings to fix any issues
spack install the_package_you_want
```

Verify compiler setup

```
[ppomorsk@l3.nibi ~]$ spack compiler find
==> Found no new compilers
==> Compilers are defined in the following files:
    /home/ppomorsk/.spack/packages.yaml
[ppomorsk@l3.nibi ~]$ spack compiler list
==> Available compilers
-- gcc almalinux9-x86_64 ----------------------------------------
[e]  gcc@14.3.0  [e]  gcc@13.3.0  [e]  gcc@12.3.1  [e]  gcc@11.5.0  [e]  gcc@11.4.1

-- llvm-amdgpu almalinux9-x86_64 --------------------------------
[e]  llvm-amdgpu@6.4.1
[ppomorsk@l3.nibi ~]$ spack compiler info  gcc@14.3.0
gcc@=14.3.0 languages:='c,c++,fortran' arch=linux-almalinux9-x86_64:
  prefix: /cvmfs/soft.computecanada.ca/gentoo/2023/x86-64-v3/usr
  compilers:
    c: /cvmfs/soft.computecanada.ca/gentoo/2023/x86-64-v3/usr/bin/gcc-14
    cxx: /cvmfs/soft.computecanada.ca/gentoo/2023/x86-64-v3/usr/bin/g++-14
    fortran: /cvmfs/soft.computecanada.ca/gentoo/2023/x86-64-v3/usr/bin/gfortran-14
```

Spack storage on clusters

Spack is hungry for storage and gives unintuitive errors if you run out

Make sure you have enough storage, for spack root and tmp

Put spack install directory on /project, never on /home

Do not use default /tmp on cluster login node, instead set $TMP to use /scratch

Do regular cleanups to reduce disk usage

Remove no longer need build packages if done building

```
spack clean --stage
spack gc
```

Spack caching

Spack builds can take a long time, and consume a lot of disk space

If multiple users are building and storing the same Spack packages, that will be wasteful

Spack can avoid by allowing one installation to get binary files from another if they are available

File duplication less of an issue on VAST systems with deduplication (Nibi, Trillium)

If there is sufficient need, the Alliance staff could maintain a central Spack repository containing some commonly used packages that user Spack installations could then pull from

We want to hear from users

Software installation is a complex problem, we are open to user feedback

Let us know if you need more support for Spack or other build options

Questions?