Software stack deployment for Earth System Modelling

Sergey Kosukhin
Max Planck Institute for Meteorology

IS-ENES2 Workshop on Workflow Solutions and Metadata Generation

29 September 2016
Lisbon, Portugal

Different users want different things

- Single installation of a particular model version (summer school).
- Several installations of the same model with different compilers, libraries, their versions and options (benchmarking/profiling).
- Continuous changes of source code with preferred toolchain (development).

Keep installation process simple and flexible: the less users now about software, the less decisions they want to make and vice versa.

Environments are very different

- Single machine or large-scale HPC site?
- Build everything from scratch or use provided system software?
- Which compiler? Which prerequisite packages and their versions?

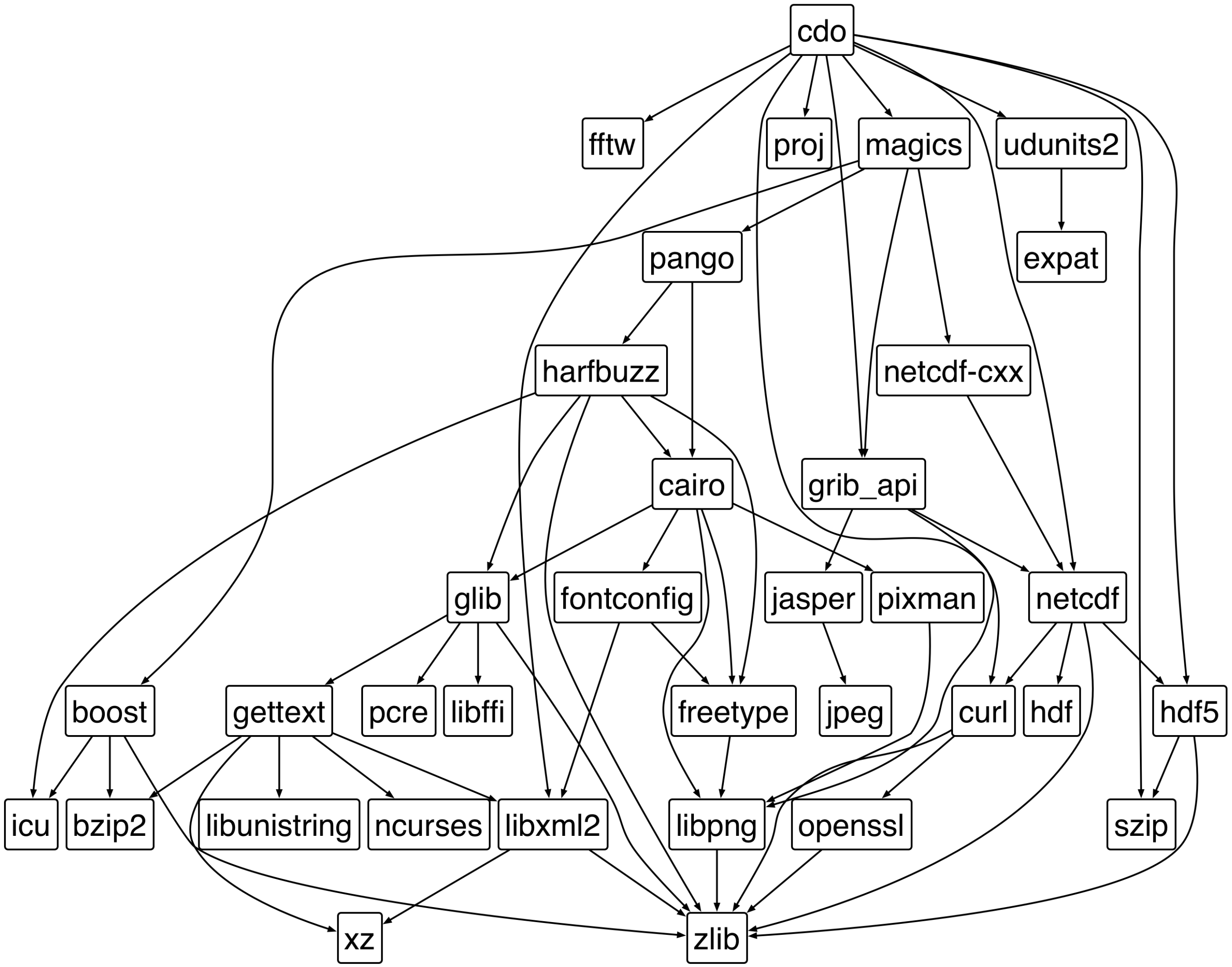
There isn't a standard way to build!

Compilers x Prerequisites x Versions x

x Options = **So many "wrong" ways to install a single package**

We don't always know in advance which one is right!

Dependency tree of the CDOs



Existing tools

- Binary package managers
  - Designed to manage a single, stable and well tested stack.
  - Install one version of each package in a single prefix (/usr).
- Port systems
  - Macports, Homebrew, Gentoo, etc.
  - Minimal support for builds parameterized by compilers, dependency versions.
- Virtual Machines and Linux Containers (Docker)
  - Containers allow users to build environments for different applications.
  - Does not solve the build problem (someone has to build the image)
  - Performance, security, and upgrade issues prevent widespread HPC deployment.

Spack is a flexible package manager

Lawrence Livermore National Laboratory

- How to install Spack:

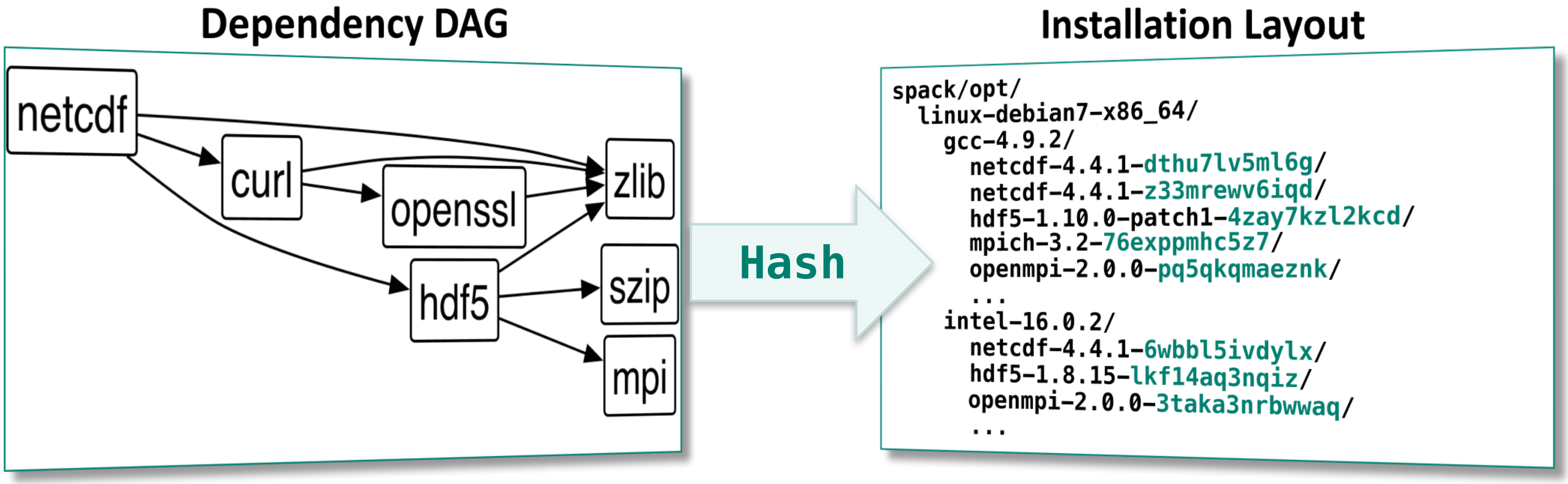
```
Get from git repository:
$ git clone https://github.com/LLNL/spack.git

Or download the archive and unzip it:
$ wget https://github.com/LLNL/spack/archive/develop.zip
$ unzip develop.zip

Setup environmental variables:
$ . ./spack/share/spack/setup-env.sh
```

- How to install a package:

```
$ spack install hdf5
```

Spack will detect available compilers and install HDF5 with all its dependencies.

Combinatorial complexity



```
spack/opt/
  linux-debian7-x86_64/
    gcc-4.9.2/
      netcdf-4.4.1-dthu7lv5ml6g/
      netcdf-4.4.1-z33mrewv6iqd/
      hdf5-1.10.0-patch1-4zay7kzl2kcd/
      mpich-3.2-76exppmhc5z7/
      openmpi-2.0.0-pq5qkqmaeznk/
      ...
    intel-16.0.2/
      netcdf-4.4.1-6wbbl5ivdylx/
      hdf5-1.8.15-lkf14aq3nqiz/
      openmpi-2.0.0-3taka3nrbwwaq/
      ...
```

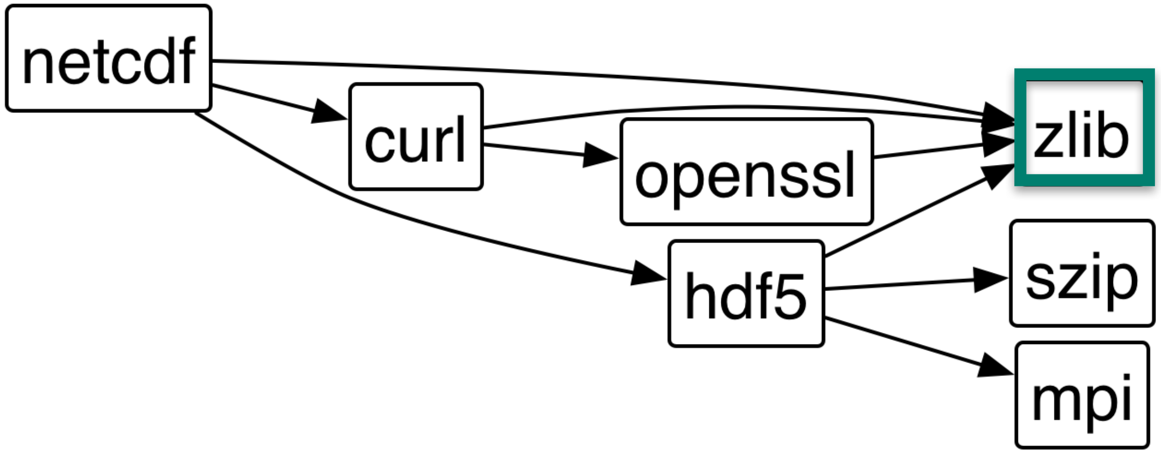
- Each unique dependency graph is a unique configuration.
- Each configuration installed in a unique directory:
  - Configurations of the same package can coexist.
- **Hash** of entire directed acyclic graph (DAG) is appended to each prefix.
- Installed packages automatically find dependencies:
  - Spack embeds RPATHs in binaries;
  - No need to use modules or set LD_LIBRARY_PATH.

Spack provides a spec syntax to describe customized DAG configurations

```
$ spack install cdo                                      unconstrained
$ spack install cdo@1.7.2                                @ custom version
$ spack install cdo@1.7.2 %gcc@4.9.2                     % custom compiler
$ spack install cdo@1.7.2 %gcc@4.9.2 +grib_api           +/~ build option
$ spack install cdo@1.7.2 os=SuSE11                      os=<frontend OS>
$ spack install cdo@1.7.2 os=CNL10                       os=<backend OS>
$ spack install cdo@1.7.2 os=CNL10 target=haswell        target=<cpu target>
```

- Each expression is a ***spec*** for a particular configuration
  - Each clause adds a constraint to the spec
  - Constraints are optional – specify only what you need.
  - Customize install on the command line!
- Syntax abstracts details in the common case
  - Makes parameterization by version, compiler, and options easy when necessary

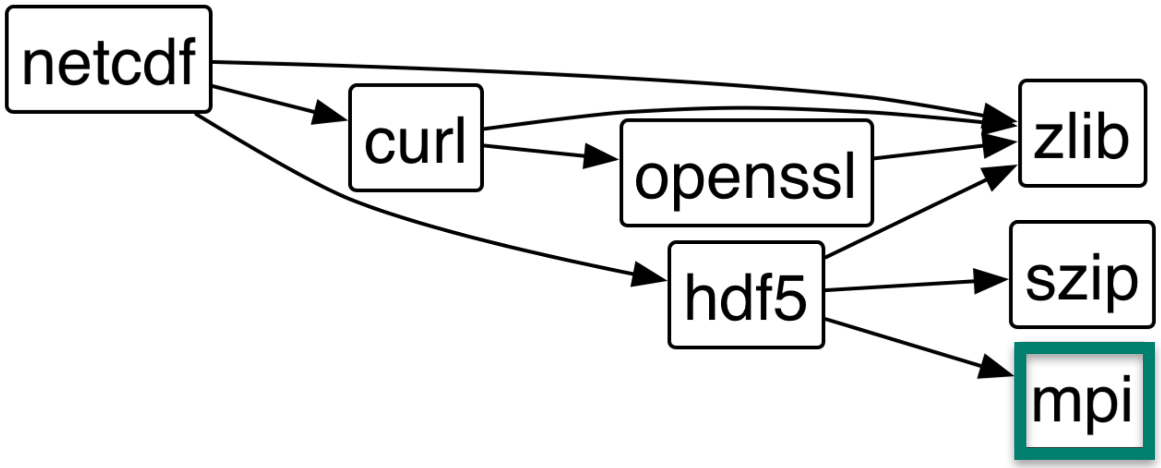
Spack specs can constrain versions of dependencies



```
$ spack install netcdf %intel@16.0.2 ^zlib@1.2.8
```

- Spack ensures one configuration of each library per DAG
- Spack can ensure that builds use the same compiler

Spack handles ABI-incompatible, versioned interfaces like MPI



- mpi is a *virtual dependency*
- Install the same package built with two different MPI implementations:

```
$ spack install netcdf ^mvapich@1.9
```

```
$ spack install netcdf ^openmpi@1.4:
```

- Let Spack choose MPI version, as long as it provides MPI 2 interface:

```
$ spack install netcdf ^mpi@2
```

Spacks allows optional dependencies

- The user can define named variants (flags):

```
   variant("python", default=False, “Build with python support”)
   depends_on("python", when="+python")
```

- And use them to install:

```
$ spack install vim +python
$ spack install vim –python
```

- Dependencies may be optional according to other conditions:

  e.g., gcc dependency on mpc from 4.5 on:

```
     depends_on("mpc", when="@4.5:")
```

Spack packages are Python scripts

```python
from spack import *

class Magics(Package):
    homepage = "https://software.ecmwf.int/wiki/display/MAGP/Magics"
    url = "https://software.ecmwf.int/wiki/download/attachments/3473464/Magics-2.29.0-Source.tar.gz"
```

Metadata

```python
    version('2.29.4', '91c561f413316fb665b3bb563f3878d1')
    version('2.29.0', 'db20a4d3c51a2da5657c31ae3de59709', preferred=True)
```

Versions

```python
    patch('no_hardcoded_python.patch')
    patch('resolve_isnan_ambiguity.patch', when='@2.29.0')
```

Patches

```python
    variant('bufr', default=False, description='Enable BUFR support')
    # More variants here...
```

Variants

```python
    depends_on('grib_api')
    # More dependencies here...
    depends_on('libemos', when='+bufr')
```

Dependencies

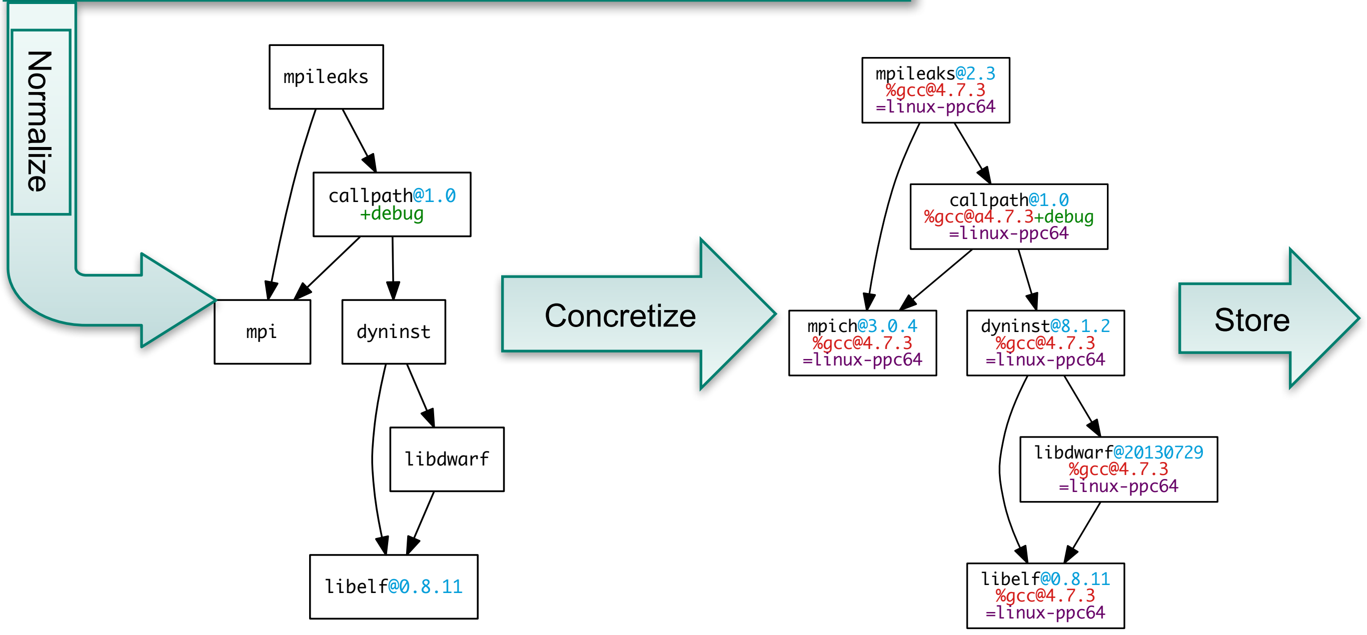
```python
    def install(self, spec, prefix):
        options = []
        options.extend(std_cmake_args)
        options.append('-DGRIB_API_PATH=%s' % spec['grib_api'].prefix)

        if '+bufr' in spec:
            options.append('-DENABLE_BUFR=ON')
            options.append('-DLIBEMOS_PATH=%s' % spec['libemos'].prefix)
        else:
            options.append('-DENABLE_BUFR=OFF')

        with working_dir('spack-build', create=True):
            cmake('..', *options)
            make()
            make('install')
```

Commands for installation

Concretization fills in missing configuration details when the user is not explicit.

```
mpileaks ^callpath@1.0+debug ^libelf@0.8.11
```

User input: *abstract* spec with some constraints



Normalize
mpileaks
callpath@1.0 +debug
mpi
dyninst
libdwarf
libelf@0.8.11
Concretize
mpileaks@2.3 %gcc@4.7.3 =linux-ppc64
callpath@1.0 %gcc@4.7.3+debug =linux-ppc64
mpich@3.0.4 %gcc@4.7.3 =linux-ppc64
dyninst@8.1.2 %gcc@4.7.3 =linux-ppc64
libdwarf@20130729 %gcc@4.7.3 =linux-ppc64
libelf@0.8.11 %gcc@4.7.3 =linux-ppc64
Store

Abstract, normalized spec with some dependencies.

Concrete spec is fully constrained and can be passed to install.

spec.yaml

```
spec:
- mpileaks:
    arch: linux-x86_64
    compiler:
      name: gcc
      version: 4.9.2
    dependencies:
      adept-utils: kszrtkpbzac3ss2ixcjkcorlaybnptp4
      callpath: bah5f4h4d2n47mgycej2mtrnrivvxy77
      mpich: aa4ar6ifj23yijqmdabeakpejcli72t3
    hash: 33hjjhxi7p6gyzn5ptgyes7sghyprujh
    variants: {}
    version: '1.0'
- adept-utils:
    arch: linux-x86_64
    compiler:
      name: gcc
      version: 4.9.2
    dependencies:
      boost: teesjv7ehpe5ksspjim5dk43a7qnowlq
      mpich: aa4ar6ifj23yijqmdabeakpejcli72t3
    hash: kszrtkpbzac3ss2ixcjkcorlaybnptp4
    variants: {}
    version: 1.0.1
- boost:
    arch: linux-x86_64
    compiler:
      name: gcc
      version: 4.9.2
    dependencies: {}
    hash: teesjv7ehpe5ksspjim5dk43a7qnowlq
    variants: {}
    version: 1.59.0
...
```

Detailed provenance is stored with the installed package

`spack find` shows what is installed

```
$ spack find
==> 54 installed packages.
-- linux-x86_64 / gcc@4.4.7 --------------------------------
ImageMagick@6.8.9-10  glib@2.42.1        libtiff@4.0.3
SAMRAI@3.9.1          graphlib@2.0.0     libtool@2.4.2
adept-utils@1.0       gtkplus@2.24.25    libxcb@1.11
atk@2.14.0            harfbuzz@0.9.37    libxml2@2.9.2
boost@1.55.0          hdf5@1.8.13        llvm@3.0
cairo@1.14.0          icu@54.1           metis@5.1.0
callpath@1.0.2        jpeg@9a            mpich@3.0.4
dyninst@8.1.2         libdwarf@20130729  ncurses@5.9
dyninst@8.1.2         libelf@0.8.13      ocr@2015-02-16
fontconfig@2.11.1     libffi@3.1         openssl@1.0.1h
freetype@2.5.3        libmng@2.0.2       otf@1.12.5salmon
gdk-pixbuf@2.31.2     libpng@1.6.16      otf2@1.4

-- linux-x86_64 / gcc@4.8.2 --------------------------------
adept-utils@1.0.1  boost@1.55.0  cmake@5.6-special
adept-utils@1.0.1  cmake@5.6     dyninst@8.1.2

-- linux-x86_64 / intel@14.0.2 -----------------------------
hwloc@1.9  mpich@3.0.4  starpu@1.1.4

-- linux-x86_64 / intel@15.0.0 -----------------------------
adept-utils@1.0.1  boost@1.55.0  libdwarf@20130729

-- linux-x86_64 / intel@15.0.1 -----------------------------
adept-utils@1.0.1  callpath@1.0.2  libdwarf@20130729
boost@1.55.0       hwloc@1.9       libelf@0.8.13
```

- All the versions coexist!

  Multiple versions of same package are ok.
- Packages are installed to automatically find correct dependencies.
- Binaries work regardless of user's environment.
- Spack also generates module files.

  Don't have to use them.

Adding compiler flags (for compiler parameter studies)

```
$ spack install ncl cflags=\'-O3 –g –fast –fpack-struct\'
```

- This would install NCL with the specified flags
  - Flags are injected via Spack's compiler wrappers.
- Flags are propagated to dependencies automatically
  - Flags are included in the **DAG hash**
  - Each build is considered a **different version**
- This provides an easy harness for doing parameter studies for tuning codes

Status of the integration of the ESM applications into Spack

- CDO (https://code.zmaw.de/projects/cdo): fully featured, main branch
- Magics (https://software.ecmwf.int/wiki/display/MAGP/Magics): almost fully featured (no python support yet), main branch
- Emoslib (https://software.ecmwf.int/wiki/display/EMOS/Emoslib): minimum implementation, main branch
- NCAR NCL (https://www.ncl.ucar.edu): WIP

Plans on ES models integration:

- MPI-ESM1
- EC-EARTH
- ICON
- FAMOUS

This work is supported by the ESiWACE project funded by the European Union's Horizon 2020 Research and Innovation Programme

Known issues

- Sometimes spack fails to prevent packages from linking against external libraries.
- Not all the packages are at the production level.
- Spack is under the continuous development without explicit stable version.
- URL lists of packages require maintenance.

Conclusion

- There is no magic: the more standard the original building system of a package is the easier it is to integrate it into spack and vice versa.
- Among other things it is a good way to document the installation process.

References

- T. Gamblin, M. LeGendre, M. R. Collette, G. L. Lee, A. Moody, B. R. de Supinski, and S. Futral. The Spack package manager: Bringing order to HPC software chaos. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 40:1–40:12, 2015.
- T. Gamblin et al. The Spack package manager: Bringing order to HPC software chaos [PDF slides]. Retrieved from https://tgamblin.github.io/files/Gamblin-Spack-SC15-Talk.pdf
- T. Gamblin. Spack: a tool for managing HPC software [PowerPoint slides]. Personal correspondence.
- Spack documentation. http://spack.readthedocs.io/en/latest

Thank you!

Sergey Kosukhin

sergey.kosukhin@mpimet.mpg.de