

Minutes-to-Modules: Rapid, Reproducible Software Packaging with Spack and CI for Research HPC on Gadi and Setonix

Justin Kin Jun Hew^{1,2}, Lawrence Bird¹, Mike Tetley¹, Harshula Jayasuriya¹, Aidan Heerdegen¹,
Felicity McCormack²

¹ Australia's Climate Simulator (ACCESS-NRI), Australian National University, Canberra, ACT 2601,
Australia

² Securing Antarctica's Environmental Future, School of Earth, Atmosphere and Environment,
Monash University, Clayton, Kulin Nations, VIC, Australia

Motivation: Why This Matters

- HPC systems evolve rapidly (compilers, MPI, libraries)
- Scientific codes depend on dozens of fragile dependencies
- Traditional module systems can't handle variant complexity
- We needed reproducible, portable, testable builds

Goal: Achieve reproducible, modular builds in minutes.

Enter Spack

- A package manager for HPC – like Conda, but built for supercomputers
- Declarative environments for consistent builds
- Fine-grained control over compilers, MPI, variants
- Binary caches enable rapid rebuilds



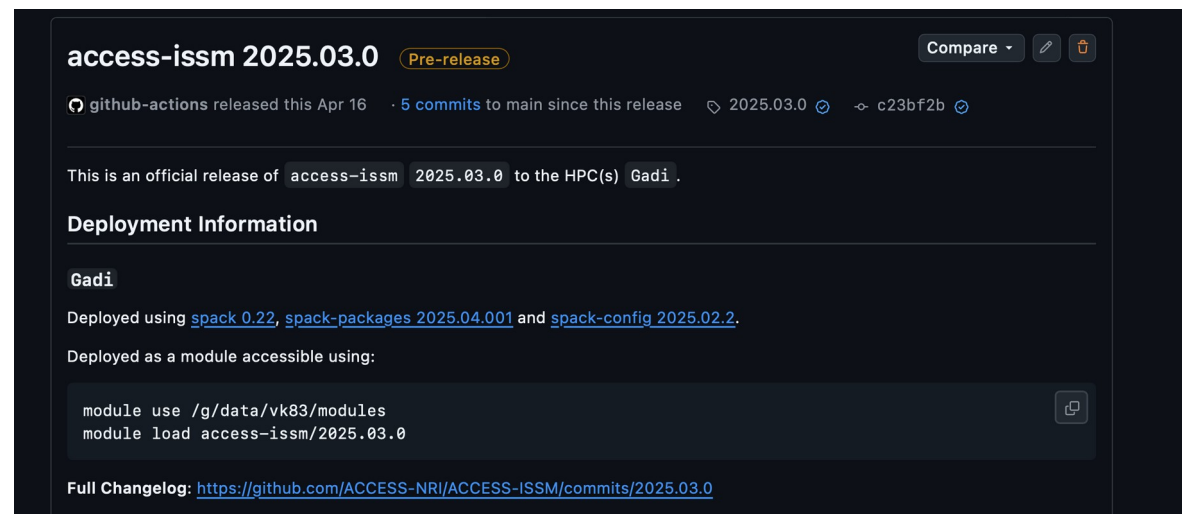
Spack

Minutes to Modules: CI Workflow

- 1. Spack environment YAML defines reproducible stack
- 2. CI builds and tests packages on Gadi
- 3. Artifacts uploaded to binary cache
- 4. Export as HPC modules under `/g/data/vk83/modules`
- → 'module load access-issm' instantly ready

Case Study: ACCESS-ISSM

- Full Spack-based build of ISSM and dependencies
- Variants: +ad, +wrappers, +cuda, +debug
- CI pipeline triggers rebuilds automatically
- Reduced build time: 6 hrs → <20 mins (cached)
- Works across Gadi and Setonix



The screenshot shows a GitHub release page for the project 'access-issm' at version '2025.03.0'. The release is marked as a 'Pre-release'. It was released by 'github-actions' on April 16, 2025, and includes 5 commits to the main branch since the previous release. The release is for the HPC(s) 'Gadi'. The deployment information section indicates that the release was deployed using 'spack 0.22', 'spack-packages 2025.04.001', and 'spack-config 2025.02.2'. It also provides the command to load the module on Gadi: 'module use /g/data/vk83/modules' followed by 'module load access-issm/2025.03.0'. A full changelog link is provided at the bottom: 'https://github.com/ACCESS-NRI/ACCESS-ISSM/commits/2025.03.0'.

access-issm 2025.03.0 Pre-release Compare

github-actions released this Apr 16 · 5 commits to main since this release · 2025.03.0 · c23bf2b

This is an official release of `access-issm` `2025.03.0` to the HPC(s) `Gadi`.

Deployment Information

Gadi

Deployed using [spack 0.22](#), [spack-packages 2025.04.001](#) and [spack-config 2025.02.2](#).

Deployed as a module accessible using:

```
module use /g/data/vk83/modules
module load access-issm/2025.03.0
```

Full Changelog: <https://github.com/ACCESS-NRI/ACCESS-ISSM/commits/2025.03.0>

Why does my ice sheet melt on login?

- Module-mess & ‘works-on-my-machine’ crashes
- PETSc update broke jobs --> 48 h queue time lost
- Cluster \neq Laptop: compilers, MPI, paths, flags
- Goal \rightarrow identical ISSM builds everywhere



Real-life picture of ice shelf calving, or... ISSM build failure

Spack turns fragile scripts into a reproducible, hash-pinned build config.

ISSM Dependency Avalanche

Dependency stack (PETSc, MUMPS, METIS, NetCDF, ...)

- 12 + libs, 3 languages, CPU vs GPU variants
- ABIs/versions shift ⇒ build fails
- Manual scripts = yak shaving

```
...# Dependencies
...# -----
...# Build-time tools
...depends_on("autoconf", type="build")
...depends_on("automake", type="build")
...depends_on("libtool", type="build")
...depends_on("m4", type="build")

...# Core build + runtime deps
...depends_on("mpi")

...# Linear-algebra stack -- only for the *non-AD* flavour
...depends_on("petsc~examples+metis+mumps+scalapack", when=~ad")
...depends_on("parmetis")
...depends_on("metis")
...depends_on("mumps~openmp", when=~openmp")
...depends_on("mumps+openmp", when="+openmp")
...depends_on("scalapack")

...# Note: ISSM's MUMPS support is not compatible with the Spack-provided
...# MUMPS, so we use the one provided by the ISSM team.

...# Optimiser
...depends_on("m1qn3")

...# Automatic-differentiation libraries
...depends_on("codipack", when="+ad")
...depends_on("medipack", when="+ad")

...# Optional extras controlled by +wrappers
...depends_on("access-triangle", when="+wrappers")
...depends_on("python@3.9.2:", when="+wrappers", type=("build", "run"))
...depends_on("py-numpy", when="+wrappers", type=("build", "run"))
```

The ISSM Package & Key Variants

- +examples (installing tutorial files)
- +ad (automatic differentiation)
- +wrappers (Python and numpy wrappers, brings access-triangle path)
- Recipe maintained in ACCESS-NRI fork

```
variant(  
    "wrappers",  
    default=False,  
    description="Enable building ISSM Python/C wrappers",  
)  
  
variant(  
    "examples",  
    default=False,  
    description="Install the examples tree under <prefix>/examples",  
)  
  
variant(  
    "ad",  
    default=False,  
    description="Build with CoDiPack automatic differentiation (drops PETSc)",  
)  
  
variant(  
    "openmp",  
    default=True,  
    description="Propagate OpenMP flags so threaded deps link cleanly",  
)
```

Feature: Automatic Differentiation

- Why AD? Sensitivity studies, inverse problems, adjoints
- Forward-mode via operator overloading (CoDiPack)
- Memory-efficient taping (MediPack)
- Replaces hand-written Jacobians; integrates into ISSM solver

CoDiPack & MediPack in ISSM

- CoDiPack: Header-only C++ template AD
- MediPack: Tape storage abstraction
- ISSM build flags: -DCODI_ForcedInlines -O3 -g
- No PETSc linkage – uses bespoke linear algebra path

```
..# -----  
..# Build environment -- inject AD and/or OpenMP compiler flags when needed  
..# -----  
..def setup_build_environment(self, env):  
..    # OpenMP support  
..    if "+openmp" in self.spec:  
..        for var in ("CFLAGS", "CXXFLAGS", "FFLAGS", "LDFLAGS"):  
..            env.append_flags(var, self.compiler.openmp_flag)  
  
..    # Automatic Differentiation extras  
..    if "+ad" in self.spec:  
..        # CoDiPack's performance tips: force inlining & keep full symbols  
..        env.append_flags(  
..            "CXXFLAGS",  
..            f"-g -O3 -fPIC {self.compiler.cxx11_flag} -DCODI_ForcedInlines", # https://issm.ess.uci.edu/trac/issm/wiki/totten#InstallingISSMwithCoDiPackAD  
..        )
```

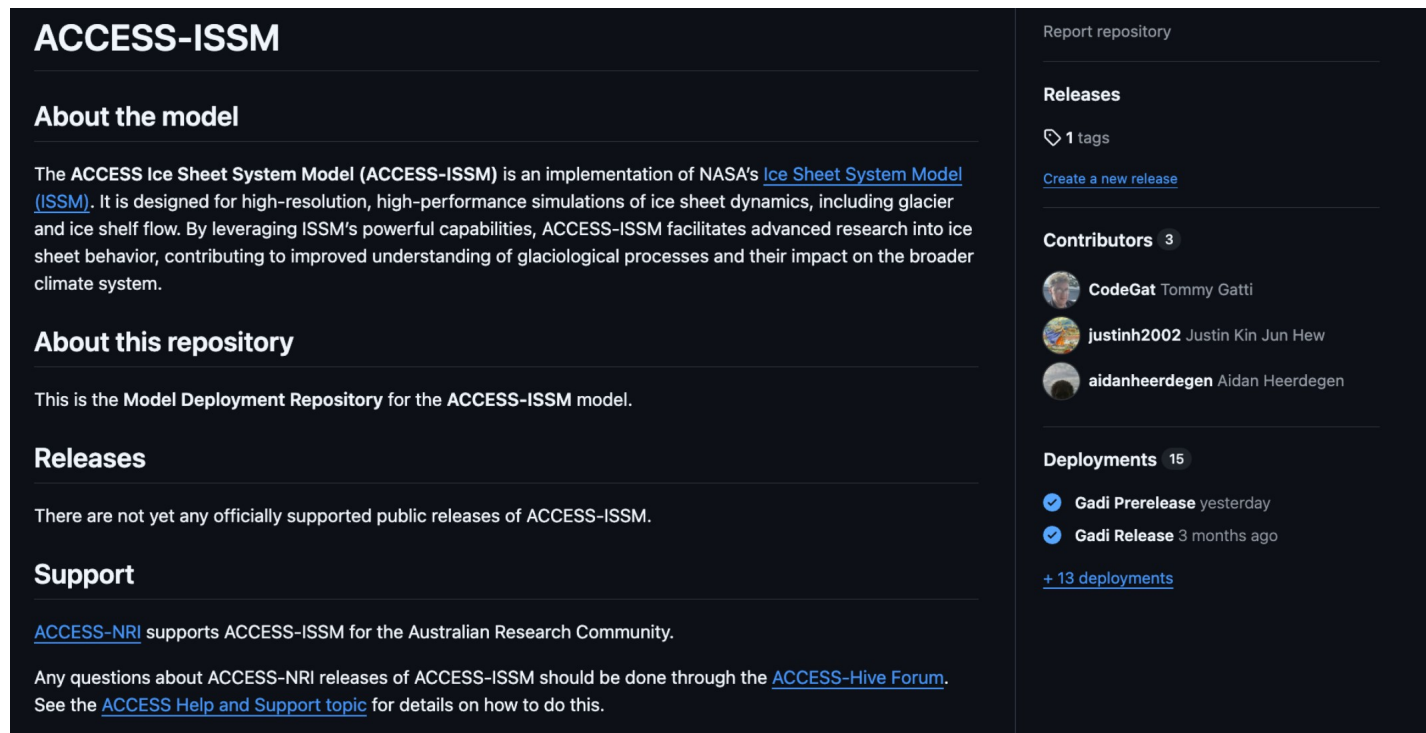
OpenMP: Threading the Solver

- +openmp injects compiler flags system-wide
- Builds MUMPS with OpenMP support when in Classic mode
- In AD mode, improves nonlinear residual assembly speed
- Disable with ~openmp if threading conflicts with MPI pinning

```
... # OpenMP support
... if "+openmp" in self.spec:
...     for var in ("CFLAGS", "CXXFLAGS", "FFLAGS", "LDFLAGS"):
...         env.append_flags(var, self.compiler.openmp_flag)
```

Environments & Lockfiles

- `spack env create -d issm-env`
- `spack add issm@access-development +cuda cuda_arch=90 +wrappers %gcc@13.2 target=x86_64`
- `spack concretize && spack install`
- `spack lock add -f spack.lock` → share & reuse



ACCESS-ISSM

About the model

The **ACCESS Ice Sheet System Model (ACCESS-ISSM)** is an implementation of NASA's [Ice Sheet System Model \(ISSM\)](#). It is designed for high-resolution, high-performance simulations of ice sheet dynamics, including glacier and ice shelf flow. By leveraging ISSM's powerful capabilities, ACCESS-ISSM facilitates advanced research into ice sheet behavior, contributing to improved understanding of glaciological processes and their impact on the broader climate system.

About this repository

This is the **Model Deployment Repository** for the **ACCESS-ISSM** model.

Releases

There are not yet any officially supported public releases of ACCESS-ISSM.

Support

[ACCESS-NRI](#) supports ACCESS-ISSM for the Australian Research Community.

Any questions about ACCESS-NRI releases of ACCESS-ISSM should be done through the [ACCESS-Hive Forum](#). See the [ACCESS Help and Support topic](#) for details on how to do this.




Report repository

Releases



1 tags

[Create a new release](#)

Contributors 3

-  **CodeGat** Tommy Gatti
-  **justinh2002** Justin Kin Jun Hew
-  **aidanheerdegen** Aidan Heerdegen

Deployments 15

-  **Gadi Prerelease** yesterday
-  **Gadi Release** 3 months ago

[+ 13 deployments](#)

ACCESS-ISSM and Spack Bundle Package

- Containerised via a `spack.yaml` with versions fixed
- Alpha release and AD pre-release

```
from spack.package import BundlePackage, maintainers, version, depends_on

class AccessIssm(BundlePackage):
    """
    .. ACCESS-ISSM bundle containing the issm Package.

    .. This is used to version the entirety of a released deployment, including
    .. the package, it's dependencies, and the version of
    .. spack-packages/spack-config that it is bundled with
    .. """

    .. homepage = "https://www.access-nri.org.au"
    .. git = "https://github.com/ACCESS-NRI/ACCESS-ISSM.git"

    .. maintainers("harshula", "justinh2002")

    .. version("latest")

    .. depends_on("issm")
```

How to run ISSM Documentation

Run ACCESS-ISSM

1 About

ACCESS-ISSM couples the **Ice Sheet System Model (ISSM)** to the ACCESS infrastructure, enabling fully parallel Antarctic and Greenland ice-sheet simulations on the [NCI Gadi supercomputer](#)[Ⓒ].

It is maintained and supported by **ACCESS-NRI**.

A high-level description of model components—including the ISSM core, pre-processing utilities, climate forcings, and coupling hooks—is available in the [ACCESS-ISSM overview.ng](#).

The example below reproduces the *MISMIP+* benchmark. Adjust the variables to suit your experiment.

On this page

1 About

2 Prerequisites

3 Quick-start (Gadi login node)

4 Running from ARE Desktop

5 File structure

experiments/mismip/ | —
access-issm/ # git clone (alpha
branch) | —

examples/mismip/ | —
mismip_driver.py # full driver
script (all steps) | —
run_pps.sh # helper: submits
PPS job | —

run_mismip_first.sh # helper:
submits early RUN job (steps
1-7) | — run_pps.sh #
copied helper – edit
walltime/resources here
| — run_mismip_first.sh #
copied helper – edit
nodes/time as needed

6 Editing job scripts

| ACCESS-ISSM configuration

7 Troubleshooting

Get help

Portability: Gadi vs Setonix

- Unified builds across architectures.
- OpenMPI vs Cray MPICH
- Same spack.yaml & binary cache.

Reproducibility & Portability

- Builds include compiler + dependency provenance
- Same environment reproducible across systems
- Aligns with FAIR software principles
- Enables portable Antarctic ice-sheet simulations

Closing Thoughts

- Spack transformed packaging on Australian HPC:
- Minutes instead of hours
- Reproducible, sharable builds
- Sustainable research software for the future

Thank you!

Questions?

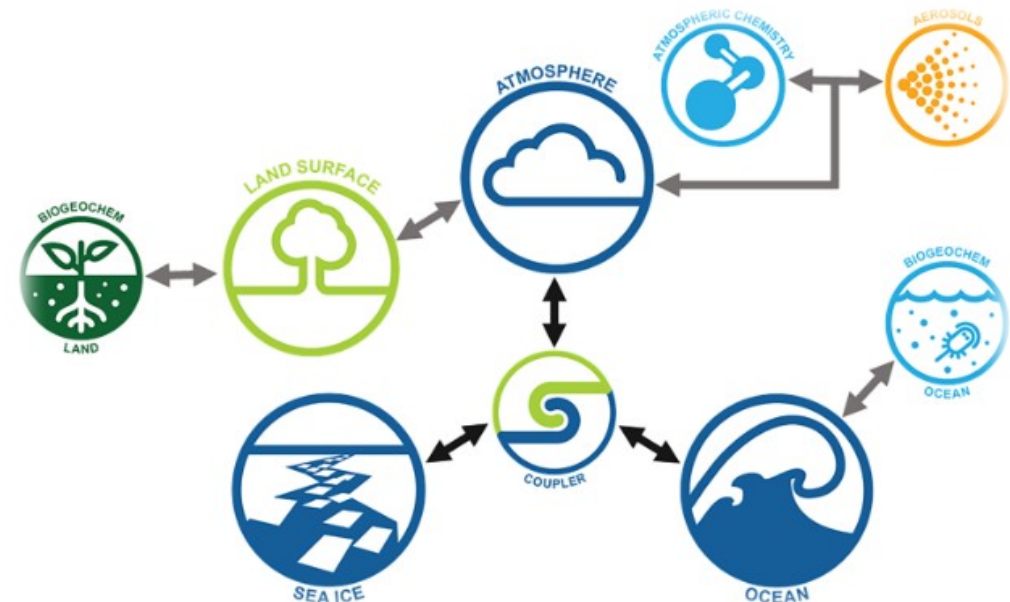


Australia's climate simulator

A decorative horizontal bar consisting of a blue segment, a light blue segment, a dark blue segment, a green segment, and three small green dots.

Brief Background – ACCESS Models

- ACCESS Models are climate models linking together mathematical representations of major Earth System components
- Complex – some models total 1.5m+ lines of code, with many dependencies!
- Scale – We support 7 models currently, 10+ by the end of the year!
- We have two main audiences for these:
 - Researchers using the models
 - Developers updating the models



Motivation

- Researchers spend days compiling dependencies.
- Inconsistent environments across Gadi and Setonix.
- Reproducibility and collaboration often break down.
- Goal: Achieve reproducible, modular builds in minutes.

The HPC Challenge

- Manual builds → inconsistent results
- Module collisions → ABI issues
- Cluster differences → MPI/compiler drift
- No provenance → irreproducible results

Enter Spack

- Open-source HPC package manager (LLNL)
- Dependency resolution & compiler-agnostic
- Reproducible spec system

- Example:
- `spack install issm +ad +wrappers %gcc@13.2 ^petsc@3.20 +cuda`

Minutes-to-Modules Pipeline

- Automate Spack via CI/CD:
 - 1. Define environment (spack.yaml)
 - 2. Cache binaries via buildcache
 - 3. Trigger CI builds (GitHub → Gadi/Setonix)
 - 4. Export as HPC module

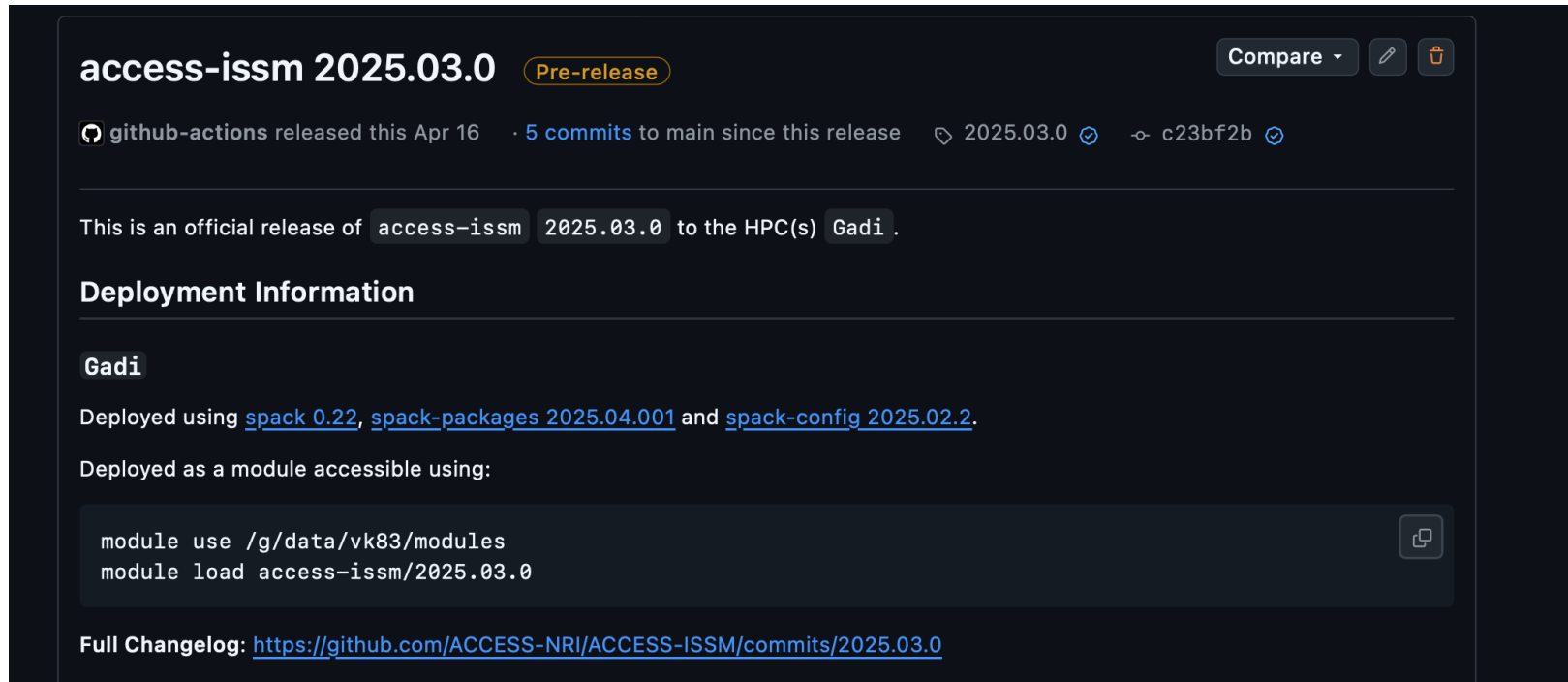
- Commit → Build → Module → Load

From Spec to Module

- `spack.yaml` → concretized `spec.yaml` → build job → install tree → modulefile
- Tracks provenance, auto-generates modules, shared via cache/repo.

Case Study: ACCESS-ISSM

- ISSM, PETSc, NetCDF/HDF5, Python stack, CI all built reproducibly with Spack.
- GitHub Actions + PBS runners ensure automation.



The screenshot shows a GitHub release page for the project 'access-issm'. The release is titled 'access-issm 2025.03.0' and is marked as a 'Pre-release'. It was released by 'github-actions' on April 16. The release includes 5 commits to the main branch since the previous release, with a commit hash of 'c23bf2b'. The release is an official release of 'access-issm 2025.03.0' to the HPC(s) 'Gadi'. The deployment information section indicates that the release was deployed using 'spack 0.22', 'spack-packages 2025.04.001', and 'spack-config 2025.02.2'. The deployment is accessible as a module on the 'Gadi' system, with the following commands: 'module use /g/data/vk83/modules' and 'module load access-issm/2025.03.0'. A full changelog is provided at the bottom of the page, with a link to 'https://github.com/ACCESS-NRI/ACCESS-ISSM/commits/2025.03.0'.

access-issm 2025.03.0 Pre-release Compare

github-actions released this Apr 16 · 5 commits to main since this release · 2025.03.0 · c23bf2b

This is an official release of `access-issm 2025.03.0` to the HPC(s) `Gadi`.

Deployment Information

Gadi

Deployed using [spack 0.22](#), [spack-packages 2025.04.001](#) and [spack-config 2025.02.2](#).

Deployed as a module accessible using:

```
module use /g/data/vk83/modules
module load access-issm/2025.03.0
```

Full Changelog: <https://github.com/ACCESS-NRI/ACCESS-ISSM/commits/2025.03.0>

CI/CD Architecture

- GitHub Action → SSH Runner (Gadi)
 - - setup Spack env
 - - concretize build matrix
 - - upload buildcache
 - - notify team
- Cross-cluster reproducibility & provenance.

Performance & Provenance

- 20× faster rebuilds with cached binaries
- `spack find --json` → full dependency tree
- Environment hashes ensure reproducibility.

Portability: Gadi vs Setonix

- Unified builds across architectures.
- OpenMPI vs Cray MPICH
- CUDA vs ROCm
- Same spack.yaml & binary cache.

Lessons Learned

- Version control specs
- Treat modules as artefacts
- Automate provenance
- Binary caches = fast rebuilds
- CI runners essential.

Research Integration

- Used in ACCESS-ISSM hydrology simulations
- Also ACCESS-OM2, CABLE
- Shared environments across ACCESS models.

Demo

- `spack env activate issm-env`
- `make issm`
- `module load issm/ad/2025.10`
- `python ais_coupled_v1.py`
- <2 min → reproducible run.

Future Directions

- Apptainer/Container integration
- Shared binary cache (Gadi ↔ Setonix)
- Unified NCRIS CI runner
- FAIR metadata registry.

Key Takeaways

- 1. Reproducibility = automation
- 2. Spack + CI = consistency
- 3. One spec, many systems
- 4. Modules in minutes.

High Performance Computing (HPC) violates many of these assumptions

- **Code is typically distributed as source**
 - With exception of vendor libraries, compilers
- **Often build many variants of the same package**
 - Developers' builds may be very different
 - Many first-time builds when machines are new
- **Code is optimized for the processor and GPU**
 - Must make effective use of the hardware
 - Can make 10-100x perf difference
- **Rely heavily on system packages**
 - Need to use optimized libraries that come with machines
 - Need to use host GPU libraries and network
- **Multi-language**
 - C, C++, Fortran, Python, others all in the same ecosystem

Some Supercomputers



Oak Ridge National Lab
Power9 / NVIDIA



RIKEN
Fujitsu/ARM a64fx



Lawrence Berkeley
National Lab
AMD Zen / NVIDIA



Argonne National Lab
Intel Xeon / Xe



Oak Ridge National Lab
AMD Zen / Radeon



Lawrence Livermore
National Lab
AMD Zen / Radeon

What is Spack?



Some fairly common (but questionable) assumptions made by package managers (conda, pip, apt, etc.)

- **1:1 relationship between source code and binary (per platform)**
 - Good for reproducibility (e.g., Debian)
 - Bad for performance optimization
- **Binaries should be as portable as possible**
 - What most distributions do
 - Again, bad for performance
- **Toolchain is the same across the ecosystem**
 - One compiler, one set of runtime libraries
 - Or, no compiler (for interpreted languages)

Outside these boundaries, users are typically on their own



Australia's climate simulator

A decorative horizontal bar at the bottom right of the page, consisting of a blue segment, a light blue segment, a dark blue segment, a green segment, and three small green dots.

Australia's climate simulator

