Building MPAS-JEDI with Guix

Riha, S.

December 23, 2025

Contents

1 Abstract

These are notes on an attempt to build a moderately complex software stack with the GNU Guix package manager. As an exemplary software stack, we pick the data assimilation system JEDI for the numerical atmospheric model MPAS-atmopshere. The documentation of the JEDI project contains detailed build instructions using the Spack package manager, which we try to adapt, somewhat loosely, to the Guix framework. The result is a Scheme module containing package definitions for MPAS-JEDI and its stack of dependencies, layered on top of pre-packaged Guix applications and libraries.

2 Motivation

The overall objective is to evaluate which of the popular package managers is suitable for providing a stable development environment within a rolling release Linux distribution. The criteria upon which suitability is determined, are mostly subjective and may be particular

to the author's personal preference. In a rolling release distribution model, new software is continuously packaged and delivered by the Linux distribution as soon as new software versions are released by upstream developers. The result is an operating system with modern applications and system libraries. While this is generally desirable, it also means that the distributed development toolchains constantly change. For users who are trying to learn how to build software packages (such as the author), it can be challenging to constantly rebuild applications to adapt to changes in the underlying software stack.

Several methods exist for maintaining multiple versions/variants of a software package on a system, including virtualization and containerization using Linux namespaces. While these rely on isolation from the host system, package managers like Spack, Nix or Guix can manage multiple versions/variants of a package on the host system. Spack is used in several software projects as a means to enable portability and allow developers to quickly create a productive and consistent computing environment. The Nix package manager is a popular cross-platform package manager for Unix-like systems, and a functional language to define reproducible package builds in a declarative way. The Guix package manager implements the functional package management discipline pioneered by Nix, but uses an implementation of the Scheme programming language instead of the Nix language. While Spack seems to have been originally created for HPC or supercomputing environments, Nix and Guix seem to have been created with a focus on general-purpose computing. Nix and Guix are developed alongside Linux distributions which integrate the respective package manager as their central administration tool.

Our near-term objective is to get to know Spack, Nix and Guix, and in this note we report on a first experience with Guix. As a test we try to package MPAS-JEDI, the implementation of the JEDI data assimilation framework for a numerical atmospheric model (MPAS-atmopshere). The MPAS-JEDI project is based on a software stack that is complex enough to test Guix for practical use, and overlaps with the author's personal/professional interest. Furthermore, it has the advantage of being already packaged in Spack, and "translating" the Spack package definition is both relatively easy, and additionally provides a first learning experience for Spack.

The Joint Effort for Data assimilation Integration (JEDI) is a data assimilation framework developed by the Joint Center for Satellite Data Assimilation (JCSDA). The Model for Prediction Across Scales (MPAS) is a collaborative project for developing atmosphere, ocean and other earth-system simulation components for use in climate, regional climate and weather studies. MPAS-JEDI is the interface between the generic components of the JEDI system and the atmospheric core of the Model for Prediction Across Scales (MPAS-atmosphere or MPAS-A).

3 Scope

The goal is to provide a single file containing Guix package definitions for MPAS-JEDI (version 3.0.2.mmm) and its dependency stack, such that MPAS-JEDI can be built with a single command, assuming a reasonably recent installation of the Guix package manager. Build tests are run as part of the build system shipped with Guix. Many tests are distributed in the source code of the upstream projects. The goal here is to successfully run as many of these tests as possible. Further testing is out of scope, mostly because we do not yet have sufficient background in meteorology to formulate any meaningful further tests. We also lack knowledge about computer I/O, necessary to validate the functionality of the NetCDF stack, in particular regarding I/O on parallel filesystems typically found in HPC production environments. As such, this note may be interesting to readers with an interest in both Guix and geophysical fluid dynamics (GFD). Readers merely interested in MPAS-JEDI, without a particular interest in Guix, are advised by the MPAS-JEDI documentation to use the Spack stack as a base environment for their builds. We suspect that requests for support and contributions to the JEDI project may be rejected if they are not based on (or are fully compatible with) the Spack stack. To readers merely interested in Guix, but not GFD, we suggest to take a look at the Guix package definitions which are included in the Guix source code. As noted below, we mostly composed our package definitions from modified snippets of the file `gnu/packages/maths.scm` (see the file in the repository), which is likely relevant to any STEM related software.

4 Methods

The Guix manual documents how to write software package definitions. Guix is written in the Guile programming language, which is an implementation of Scheme. A primer for Scheme is available online. During this test, we acquired minimal understanding of the Scheme language, just enough to allow us to extract and modify snippets of the package definitions which Guix includes. The definitions in the Scheme module file `gnu/packages/maths.scm` in the Guix source code repository was particularly useful.

To test Guix, we attempt to build MPAS-JEDI following the instructions on the JEDI documentation website, but using Guix instead of Spack. The aforementioned website recommends to use MPAS-BUNDLE, providing the dependency chain necessary to build MPAS-JEDI. We attempt to build version 3.0.2 of the MPAS-BUNDLE repository, which was released on November 14, 2024. The top-level CMakeLists.txt file

lists a number of dependencies for MPAS-BUNDLE, most importantly the MPAS-atmosphere numerical model (MPAS-Model) and MPAS-JEDI. The following list includes some of the dependencies listed in the CMakeLists.txt files of MPAS-Model and MPAS-JEDI, more specifically the arguments of some of the find_package commands (in pseudo-code):

- MPAS-Model v8.2.2
  - OpenMP COMPONENTS Fortran
  - MPI REQUIRED COMPONENTS Fortran
  - NetCDF REQUIRED COMPONENTS Fortran C
  - PnetCDF REQUIRED COMPONENTS Fortran
  - PIO REQUIRED COMPONENTS Fortran C
  - if MPAS_PROFILE is enabled, the package GPTL is required
- mpas-jedi 3.0.2.mmm
  - ecbuild 3.3.2 REQUIRED
  - OpenMP COMPONENTS CXX Fortran
  - MPI REQUIRED COMPONENTS CXX Fortran
  - Boost REQUIRED
  - if OpenMP is available
    - if MPAS_DOUBLE_PRECISION is enabled
      - MPAS 8.0 REQUIRED COMPONENTS DOUBLE_PRECISION core_atmosphere OpenMP
    - if not
      - MPAS 8.0 REQUIRED COMPONENTS core_atmosphere OpenMP
    - atlas 0.35.0 REQUIRED COMPONENTS OMP OMP_Fortran
  - if not
    - if MPAS_DOUBLE_PRECISION is enabled
      - MPAS 8.0 REQUIRED COMPONENTS DOUBLE_PRECISION core_atmosphere)
    - if not
      - MPAS 8.0 REQUIRED COMPONENTS core_atmosphere)
  - atlas 0.35.0 REQUIRED
  - oops 1.10.0 REQUIRED
  - saber 1.10.0 REQUIRED
  - ioda 2.9.0 REQUIRED
  - ufo 1.10.0 REQUIRED

It may be interesting to compare the dependencies listed in the CMakeLists.txt files of MPAS-BUNDLE to those of MPAS-JEDI version 3.0.2.mmm itself. Note, for example, that the former lists git tag d772173 for the dependency ”oops”, but the latter only requires a minimum version of 1.10.0 for the ”oops” package. According to the ”oops” repository, commit d772173 was made more than 4 months (and many commits) after the release date of version 1.10.0.

The `REQUIRED` keyword indicates that the dependency is non-optional, and the `COMPONENTS` keyword is followed by required components. The version number does not have to be matched exactly (unless followed by the `EXACT` keyword), but must be compatible with the given number. The `find_package` command has two modes of operation, either *Module* or *Config* mode. All of the above `find_package` argument lists have *basic signature*, implying that the command searches in Config mode first. The MPAS-Model repository includes `Find<PackageName>.cmake` modules for NetCDF, PnetCDF, PIO and GPTL. These are imported in its CMakeLists.txt before `find_package` is called. If we understand correctly, the Module mode may, for example, be necessary if a dependency has not been built with cmake, in which case it may not provide a `<lowercasePackageName>-config.cmake` or `<PackageName>Config.cmake`.

After pinning the versions of the stack directly below MPAS-JEDI via the MPAS-BUNDLE repository, the next task is to get an idea about the lower parts of the stack. MPAS-JEDI version 3.0.2.mmm is apparently more recent than what is used in the latest tagged JEDI-BUNDLE version that is tagged as JEDI-SKYLAB 8.0 release, which is version 3.1.0.jcsda. The JEDI-BUNDLE contains not just MPAS-JEDI, but several other projects under the JEDI umbrella. Looking at the table listing compatible versions of JEDI and spack-stack, we see that JEDI-BUNDLE version skylab-v8.0.0 requires spack-stack version 1.7.0, but that from October 2024 onward, the required spack-stack version was increased. We could not find out which version of spack-stack exactly is needed, and tried using a more recent version 1.9.1, released on April 7, 2025, as a guideline for picking package versions of the lower parts of the stack. There is a package configuration file packages.yaml, which associates package names with specific versions and configuration variants. Note that these may be default variants, and that they may be overridden by subsequent commands. Of interest are, among others, the following lines:

- bufr: 12.1.0 +python
- bufr-query: 0.0.4 +python
- crtm: +fix
- ecbuild: 3.7.2
- eckit: 1.28.3 linalg=eigen,lapack compression=lz4,bzip2

- ecmwf-atlas: 0.40.0 +fckit +trans +tesselation +fftw
- fckit: 0.13.2 +eckit
- gsl-lite: 0.37.0
- hdf5: 1.14.3 +hl +fortran +mpi +threadsafe ~szip
- jedi-cmake: 1.4.0
- nccmp: 1.9.0.1
- ncio: 1.1.2
- netcdf-c: 4.9.2 +dap +mpi ~parallel-netcdf ~szip build_system=autotools
- netcdf-cxx4: 4.3.1
- netcdf-fortran: 4.6.1
- odc: 1.5.2 ~fortran
- openmpi: ~internal-hwloc +two_level_namespace
- parallelio: 2.6.2 +pnetcdf
- parallel-netcdf: 1.12.3
- udunits: 2.2.28

In the test described here, this list serves as a guideline for choosing package versions for packages which are not explicitly listed in the CMakeLists.txt of MPAS-BUNDLE, but we do not follow the guideline strictly. Note that the above list does not include the C and Fortran toolchains.

5 Results

We define 37 packages, including 3 package variants of existing Guix packages and 7 'data-packages' containing data only. The tests included in the upstream code pass with two exceptions. The package 'jcsda-ioda' contains one failing test (out of 410 total), and the package jcsda-mpas-jedi contains a failing test regarding coding norms. The cause of the latter is apparently a linter file missing from the source repository, and we interpret only the former as possibly geophysically relevant. As a disclaimer, note that no further tests or experiments have been performed, not even most basic ones.

The Cmake build recipes of several upstream projects perform automated binary data downloads during the build process. However,

Guix packages are built in an isolated container environment, which does not seem to allow networking by default during the build stage. This leads to errors. We saw a patch to Guix that seems to allow for git-lfs downloads during checkout, but could not get this to work. Even if it works, we speculate that this would only allow for downloading git-lfs files added to the source repository, and not for data files living in separate repositories. As a simple workaround for all of these cases, we include a bash script for downloading the files to local storage, and corresponding Guix 'data-packages', with the sole purpose of transferring the files from an arbitrary local storage path to the file store managed by the Guix daemon.

Initially we tried to build the package stack on top of Guix revision `dbe04d0` from November 24, 2025. However, the tests for the BUFR packages failed, and we reverted (rather randomly) to revision `c1604c4` from July 5, 2025. This revision worked, apart from the failing tests mentioned above. Guix has a feature called *inferiors*, which allows composing different Guix revisions in arbitrary ways. We have not fully understood this feature yet, and do not use it. As a consequence, the entire package stack is pinned to one single revision of Guix. The following lists selected components of the stack, including some of the new packages:

- netcdf-paral-4.9.0
- hdf4-alt-4.2.16-2
- jcsda-vader-1.6d56a1e
- netcdf-fortran-parallel-4.5.3
- hdf5-parallel-openmpi-1.14.6
- jcsda-saber-1.bba6f7e
- mpas-v8.2.2
- pnetcdf-fortran-1.13.0
- jcsda-oops-1.d772173
- ecmwf-fckit-0.13.2
- ncep-bufr-query-v0.0.4
- jcsda-mpas-jedi-3.0.2.mmm
- lapack-3.12.1
- gcc-11.4.0-lib
- udunits-2.2.28

- gfortran-toolchain-11.4.0
- ecmwf-eckit-1.28.3
- qhull-2020.2
- jcsda-ufo-1.94d50d6
- glibc-2.39
- fftw-3.3.10
- jcsda-ioda-1.d49ed17
- zlib-1.3
- ecmwf-atlas-0.40.0
- netcdf-cxx4-parallel-openmpi-4.3.1
- ecmwf-odc-1.5.2
- pnetcdf-1.13.0
- openmpi-4.1.6
- gfortran-11.4.0-lib
- python-3.11.11
- jcsda-crtm-3.73102a2
- netcdf-parallel-openmpi-4.9.0

Finally, a significant amount of memory seems to be necessary to compile the package definitions. We allocated 28 GiB of memory to the (virtual) build machine. For the ”oops” and ”ufo” packages, it was necessary to limit the number of concurrent processes of the build process, apparently to avoid uncontrolled termination due to shortage of memory.

6 Summary

In summary, we almost achieved the goal of providing a single file containing Guix package definitions for MPAS-JEDI (version 3.0.2.mmm) and its dependency stack, such that MPAS-JEDI can be built with a single command. Instead we provide two files, a package definition file and a separate bash script for downloading the data files. The files can be downloaded here, and further instructions are contained in the readme.txt file.