Building ALPS 2.3.4 from source on Ubuntu without root

micromamba, with a .deb-extraction fallback

1 September 2026

Contents

Overview

No root means `apt install` is unavailable, so every dependency must come from a user-writable prefix. The clean way is a **micromamba** (conda-forge) environment under `$HOME` — the same approach used for the rootless Rocky Linux build, and portable across Ubuntu 20.04 / 22.04 / 24.04 regardless of the system GCC and Python versions.

A secondary method — unpacking `.deb` files into a prefix — is included for locked-down machines where even micromamba's download is blocked but an apt mirror is reachable.

CMake downloads and compiles **Boost 1.87** itself. Never install `libboost-all-dev` or point ALPS at any other Boost.

Method 1 — micromamba (recommended)

1.1 Install micromamba

```
mkdir -p "$HOME/bin" "$HOME/src" && cd "$HOME/bin"
curl -L -o micromamba \
  https://github.com/mamba-org/micromamba-releases/releases/latest/download/micromamba-linux-64
chmod +x micromamba
./micromamba --version
```

1.2 Create the environment

```
export MAMBA_ROOT_PREFIX=$HOME/micromamba
$HOME/bin/micromamba create -y -n alps -c conda-forge \
    gcc_linux-64=12 gxx_linux-64=12 gfortran_linux-64=12 \
    cmake make \
    hdf5 openblas \
    openmpi \
    python=3.11 numpy scipy
```

Pin CMake below 4 (ALPS's bundled Boost super-build still needs pre-4 CMake policy support):

```
$HOME/bin/micromamba install -y -n alps -c conda-forge 'cmake>=3.24,<4'
```

1.3 Environment script

```
# ~/alps-env.sh  --  source ~/alps-env.sh
export MAMBA_ROOT_PREFIX=$HOME/micromamba
eval "$("$HOME/bin/micromamba" shell hook -s bash)"
micromamba activate alps

export ALPS_PREFIX=$HOME/opt/alps
export DEPS_PREFIX="$CONDA_PREFIX"
export CC=$(which x86_64-conda-linux-gnu-gcc)
export CXX=$(which x86_64-conda-linux-gnu-g++)
export FC=$(which x86_64-conda-linux-gnu-gfortran)

if [ -f "$ALPS_PREFIX/bin/alpsvars.sh" ]; then
    source "$ALPS_PREFIX/bin/alpsvars.sh"
    # alpsvars.sh sets PYTHONPATH=$ALPS_HOME/lib, but pyalps installs to
    # lib/python3.11/site-packages -- fix it:
    export PYTHONPATH="$ALPS_PREFIX/lib/python3.11/site-packages:$PYTHONPATH"
    export LD_LIBRARY_PATH="$ALPS_PREFIX/lib:$LD_LIBRARY_PATH"
fi
```

1.4 Configure, build, install

```
source ~/alps-env.sh
cd "$HOME/src" && git clone --depth 1 https://github.com/alpsim/ALPS alps-src

cmake -S alps-src -B alps-build \
      -DCMAKE_BUILD_TYPE=Release \
      -DCMAKE_INSTALL_PREFIX="$ALPS_PREFIX" \
      -DCMAKE_PREFIX_PATH="$CONDA_PREFIX" \
      -DPython3_EXECUTABLE="$(python -c 'import sys;print(sys.executable)')" \
      -DCMAKE_CXX_FLAGS="-DBOOST_NO_AUTO_PTR -DBOOST_FILESYSTEM_NO_CXX20_ATOMIC_REF"

cmake --build   alps-build -j8          # -j$(nproc) only on a machine that is yours
cmake --build   alps-build -t test
cmake --install alps-build
```

Confirm in the configure log that HDF5, MPI, BLAS, and Python all resolve under `$CONDA_PREFIX` and not `/usr`.

Method 2 — extract .deb files into a prefix (no package manager)

`apt-get download` fetches a package without installing it and needs no root, as long as the package is in an enabled apt source. `dpkg-deb -x` then unpacks it anywhere.

```
mkdir -p "$HOME/debs" "$HOME/opt/deps" && cd "$HOME/debs"

apt-get download \
    libhdf5-dev libhdf5-103 \
    libopenblas-dev libopenblas0-pthread \
    libopenmpi-dev libopenmpi3 openmpi-bin openmpi-common \
    libgfortran5

for f in *.deb; do dpkg-deb -x "$f" "$HOME/opt/deps"; done
```

Then set a dependency environment (the hand-built branch of the upstream guide):

```
export DEPS_PREFIX=$HOME/opt/deps
export ALPS_PREFIX=$HOME/opt/alps
export PATH=$DEPS_PREFIX/usr/bin:$PATH
export LD_LIBRARY_PATH=$DEPS_PREFIX/usr/lib/x86_64-linux-gnu:$DEPS_PREFIX/usr/lib:$LD_LIBRARY_PATH
export CMAKE_PREFIX_PATH=$DEPS_PREFIX/usr:$CMAKE_PREFIX_PATH
export HDF5_ROOT=$DEPS_PREFIX/usr
```

Caveats: no dependency resolution, and some `.deb` files hardcode `/usr` paths in `.pc`/CMake files that you must patch by hand. You still need a compiler >= GCC 10.5 — if the system `g++` is older (Ubuntu 20.04 ships GCC 9), add `apt-get download gcc-11 g++-11 gfortran-11 cpp-11 libstdc++-11-dev` and unpack those too, or fall back to Method 1. Treat this method as a last resort.

Package reference

Dependency	Ubuntu apt package (if you had sudo)	conda-forge package (Method 1)
C / C++ compiler	`build-essential`, or `gcc-11 g++-11`	`gcc_linux-64`, `gxx_linux-64`
Fortran compiler	`gfortran` (or `gfortran-11`)	`gfortran_linux-64`
Make	`make` (in `build-essential`)	`make`
CMake (>= 3.18, < 4)	`cmake`	`cmake`
Git	`git`	(system)
HDF5	`libhdf5-dev`	`hdf5`
BLAS / LAPACK	`libopenblas-dev` (or `libopenblas-openmp-dev`)	`openblas`
MPI	`libopenmpi-dev`, `openmpi-bin`	`openmpi`
Python 3.11	`python3.11`, `python3.11-dev` (or `python3`, `python3-dev`)	`python=3.11`
NumPy	`python3-numpy`	`numpy`
SciPy	`python3-scipy`	`scipy`
Boost 1.87	built by CMake — do **not** install `libboost-all-dev`	same
(optional) plotting	`python3-matplotlib`	`matplotlib`

Ubuntu release notes: 20.04 → GCC 9 (too old — use `gcc-11` or Method 1); 22.04 → GCC 11 (fine); 24.04 → GCC 13, NumPy 2 (fine, and Boost 1.87 is already what CMake fetches).

Verification

```
source ~/alps-env.sh
which spinmc                                        # ~/opt/alps/bin/spinmc
ldd "$(which spinmc)" | grep -E 'hdf5|mpi|openblas'   # resolve into the env, not /lib
python -c "import pyalps, numpy, scipy; print('ok')"
```

Physics smoke test (2-D Ising, cluster updates):

```
import pyalps
p=[{'LATTICE':'square lattice','L':16,'MODEL':'Ising','J':1,'T':T,
    'THERMALIZATION':10000,'SWEEPS':50000,'UPDATE':'cluster'} for T in (2.0,2.27,2.8)]
inp=pyalps.writeInputFiles('ising',p)
pyalps.runApplication('spinmc',inp,Tmin=2,writexml=True)
for s in pyalps.flatten(pyalps.loadMeasurements(
        pyalps.getResultFiles(prefix='ising'), '|Magnetization|')):
    print(s.props['T'], s.y[0])
# |M|: ~0.91 (T=2.0)  ->  ~0.71 (T=2.27)  ->  ~0.22 (T=2.8)
```

Ubuntu troubleshooting

`c++: error: unrecognized command line option '-std=c++17'` — system GCC is < 10.5 (Ubuntu 20.04). Use Method 1, or unpack `gcc-11`/`g++-11` and pass `-DCMAKE_C_COMPILER=gcc-11 -DCMAKE_CXX_COMPILER=g++-11 -DCMAKE_Fortran_COMPILER=gfortran-11`.

CMake found `/usr/lib/x86_64-linux-gnu/libhdf5.so` instead of your prefix — delete `alps-build`, reconfigure with `-DHDF5_ROOT=$DEPS_PREFIX/usr` (or `$CONDA_PREFIX`) and `-DHDF5_NO_FIND_PACKAGE_CONFIG_FILE=ON`.

Link errors mentioning `boost::filesystem` / `auto_ptr` — a system Boost was picked up. Ensure `libboost-all-dev` is not installed / not unpacked, no `BOOST_ROOT` is set, and the two `-D... CMAKE_CXX_FLAGS` are present.

`import pyalps` -> `ModuleNotFoundError` — the `alpsvars.sh` `PYTHONPATH` bug; add `$ALPS_PREFIX/lib/python3.11/site-` to `PYTHONPATH` (the env script above does this).

OOM during compile — lower `-j`. Boost.Python and the ALPS template-heavy units can each exceed 1 GB RAM.

Building ALPS 2.3.4 from source on Rocky Linux 8 without root

As actually performed on host `crossfire` — micromamba strategy

1 September 2026

Contents

Summary

ALPS 2.3.4 was built from source and installed entirely under `$HOME` on a Rocky Linux 8.10 workstation, with **no root access at any point**. Because the machine's system compiler and Python are too old and it has no working environment modules, every dependency was supplied by a **micromamba** (conda-forge) environment. ALPS itself was then configured with CMake, which downloads and compiles Boost 1.87 on its own.

Result:

- Build reached 100 % with zero compile errors.
- `ctest` — **162 / 162 tests passed** (81 s).
- End-to-end check: `pyalps` drove `spinmc` through a 2-D Ising run and read the results back; magnetisation curve crossed the critical point correctly.

Install footprint:

Path	Size	Keep?
`~/bin/micromamba`	17 MB	yes
`~/micromamba/envs/alps`	~2.1 GB	yes (all dependencies)
`~/opt/alps`	61 MB	yes (**ALPS install**)
`~/src/alps-src`	30 MB	optional (source)
`~/src/alps-build`	1.6 GB	**delete after install**

---

1. The machine

```
$ cat /etc/rocky-release          Rocky Linux release 8.10 (Green Obsidian)
$ gcc --version                    gcc (GCC) 8.5.0            <- too old (need >= 10.5)
$ cmake --version                  cmake version 3.26.5       <- ok
$ python3 --version                Python 3.6.8               <- too old (need >= 3.9)
$ command -v mpicc mpirun          (none)                     <- no MPI
$ ls /opt/rh/                      gcc-toolset-{11..15}       <- see note
$ module avail                     dot module-git ...         <- no compiler/lib modules
$ command -v micromamba conda      (none)                     <- no user package manager
$ df -h $HOME                      1.3 TB free                <- plenty
```

Note on `/opt/rh`: only `gcc-toolset-14` and `gcc-toolset-15` contain a real `gcc`/`g++`; 11–13 are metapackage stubs, and **none** ship `gfortran`. That rules the toolsets out for a build that needs Fortran (OpenBLAS).

System packages that *were* used (all present in a default Rocky 8 install, no installation needed): `git`, `curl`, `tar`, `bzip2`, `make` (only as a bootstrap; the build uses the conda `make`), and `python3.11` / `python3.12` from the base repo — present here but ultimately not needed once micromamba supplied its own Python.

---

2. Why the micromamba strategy

The upstream guide offers three strategies. Mapping them to this host:

Strategy	Needs	Verdict here
A — environment modules	`module avail` lists gcc/hdf5/openmpi/openblas	**No** — no such modules
B — micromamba	writable `$HOME`, outbound HTTPS	**Yes**
C — hybrid (modules + hand-built)	a usable module compiler	**No** — nothing to build on

micromamba is a single static binary and installs nothing system-wide.

---

3. Step by step

3.1 Install micromamba

```
mkdir -p "$HOME/bin" "$HOME/src"
cd "$HOME/bin"
curl -L -o micromamba \
  https://github.com/mamba-org/micromamba-releases/releases/latest/download/micromamba-linux-64
```

```
chmod +x micromamba
./micromamba --version        # 2.9.0
```

(The `micro.mamba.pm` redirect was returning HTTP 503 the day this was done; the GitHub release URL above is the reliable equivalent.)

3.2 Create the dependency environment

```
export MAMBA_ROOT_PREFIX=$HOME/micromamba
$HOME/bin/micromamba create -y -n alps -c conda-forge \
    gcc_linux-64=12 gxx_linux-64=12 gfortran_linux-64=12 \
    cmake make \
    hdf5 openblas \
    openmpi \
    python=3.11 numpy scipy
```

Exact versions that resolved:

conda-forge package	Version	Role
`gcc_linux-64`, `gxx_linux-64`, `gfortran_linux-64`	12.4.0	C / C++ / Fortran compilers
`cmake`	3.31.8 (see 3.3)	build system
`make`	4.4.1	build driver
`hdf5`	2.2.0 (`nompi`)	HDF5 C/HL libraries
`openblas` / `libopenblas`	0.3.34 (pthreads)	BLAS / LAPACK
`openmpi`	5.0.10	MPI
`python`	3.11.16	interpreter Boost.Python binds to
`numpy`	2.4.6	required by pyalps; forces Boost >= 1.87
`scipy`	1.17.1	required by pyalps

3.3 Pin CMake below 4

conda-forge resolved `cmake` 4.x first. ALPS (and its bundled Boost super-build) still declare compatibility with CMake < 3.5 policies, which CMake 4 removed. Downgrade once:

```
$HOME/bin/micromamba install -y -n alps -c conda-forge 'cmake>=3.24,<4'
# -> cmake 3.31.8
```

3.4 The environment script

Written once as `~/alps-env.sh` and sourced before every build and every run:

```
# ~/alps-env.sh  --  source this:  source ~/alps-env.sh
export MAMBA_ROOT_PREFIX=$HOME/micromamba
eval "$("$HOME/bin/micromamba" shell hook -s bash)"
micromamba activate alps

export ALPS_PREFIX=$HOME/opt/alps
export DEPS_PREFIX="$CONDA_PREFIX"
export CC=$(which x86_64-conda-linux-gnu-gcc)
export CXX=$(which x86_64-conda-linux-gnu-g++)
export FC=$(which x86_64-conda-linux-gnu-gfortran)

if [ -f "$ALPS_PREFIX/bin/alpsvars.sh" ]; then
    source "$ALPS_PREFIX/bin/alpsvars.sh"
    # alpsvars.sh points PYTHONPATH at $ALPS_HOME/lib, but pyalps installs to
```

```
    # lib/python3.11/site-packages -- correct it:
    export PYTHONPATH="$ALPS_PREFIX/lib/python3.11/site-packages:$PYTHONPATH"
    export LD_LIBRARY_PATH="$ALPS_PREFIX/lib:$LD_LIBRARY_PATH"
fi
```

3.5 Clone and configure

```
source ~/alps-env.sh
cd "$HOME/src"
git clone --depth 1 https://github.com/alpsim/ALPS alps-src

cmake -S alps-src -B alps-build \
      -DCMAKE_BUILD_TYPE=Release \
      -DCMAKE_INSTALL_PREFIX="$ALPS_PREFIX" \
      -DCMAKE_PREFIX_PATH="$CONDA_PREFIX" \
      -DPython3_EXECUTABLE="$(python -c 'import sys; print(sys.executable)')" \
      -DCMAKE_CXX_FLAGS="-DBOOST_NO_AUTO_PTR -DBOOST_FILESYSTEM_NO_CXX20_ATOMIC_REF"
```

Confirm in the configure output that every dependency resolves to the conda environment and not to `/usr`:

```
-- Python interpreter /home/rmarc/micromamba/envs/alps/bin/python3.11 (3.11.16)
-- Found MPI_CXX: .../envs/alps/lib/libmpi.so (version 3.1)
-- Found BLAS: .../envs/alps/lib/libopenblas.so
-- Found HDF5: hdf5-shared (found version "2.2.0") components: C HL
-- Downloading the most recent Boost release...   Boost Version: 1_87_0
-- numpy version 2.4.6
```

3.6 Build, test, install

```
cmake --build   alps-build -j8
cmake --build   alps-build -t test      # ctest: 162/162 passed
cmake --install alps-build              # -> ~/opt/alps
```

`-j8` (not `-j$(nproc)`): the DMFT / maxent translation units each need ~1 GB RAM under `-O3`; eight parallel on a 15 GB box is the safe ceiling.

3.7 The pyalps path fix

The generated `~/opt/alps/bin/alpsvars.sh` sets `PYTHONPATH=$ALPS_HOME/lib`, but the package actually lands at `~/opt/alps/lib/python3.11/site-packages/pyalps/`. Without the correction in `~/alps-env.sh` (§3.4), `import pyalps` fails with `ModuleNotFoundError`.

3.8 Optional: matplotlib for the tutorials

The shipped tutorial `.py` scripts plot with matplotlib, which is not a build dependency:

```
micromamba install -n alps -c conda-forge matplotlib     # 3.11.1
```

Over SSH with no X display, set `export MPLBACKEND=Agg` and replace `plt.show()` with `plt.savefig("out.png")`.

4. Verification

```
source ~/alps-env.sh
which spinmc                                    # ~/opt/alps/bin/spinmc
python -c "import pyalps, numpy, scipy; print('ok')"
```

2-D Ising, cluster updates, `L=16`, sweeps 5×10^4, via `pyalps.runApplication('spinmc', ...)`:

T	|Magnetization|
2.00	0.911 ± 0.0002
2.20	0.794 ± 0.0006
2.27	0.713 ± 0.0009
2.40	0.519 ± 0.001
2.80	0.217 ± 0.001

Order parameter collapses through T_c ≈ 2.269 — correct.

---

5. Package-name reference for Rocky Linux

We installed **no RPMs** — every dependency came from conda-forge. The table below maps each dependency to (a) the conda-forge package actually used and (b) the Rocky Linux 8 RPM you would use instead if installing with `sudo` (that route is its own document).

Dependency	conda-forge package used	Rocky 8 RPM equivalent	RPM repo
C / C++ compiler	`gcc_linux-64`, `gxx_linux-64` 12.4	`gcc-toolset-14`, `gcc-toolset-14-gcc-c++`	AppStream
Fortran compiler	`gfortran_linux-64` 12.4	`gcc-toolset-14-gcc-gfortran`	AppStream
CMake	`cmake` 3.31	`cmake`	AppStream
Make	`make` 4.4	`make`	BaseOS
HDF5	`hdf5` 2.2	`hdf5-devel`	EPEL
BLAS / LAPACK	`openblas` / `libopenblas` 0.3.34	`openblas-devel`	PowerTools/CRB
MPI	`openmpi` 5.0	`openmpi-devel` + `environment-modules`	AppStream / EP
Python	`python` 3.11.16	`python3.11`, `python3.11-devel`	AppStream
NumPy	`numpy` 2.4	`python3.11-numpy`	AppStream
SciPy	`scipy` 1.17	`python3.11-scipy`	AppStream
Boost	*(built by CMake — 1.87)*	*(do **not** install `boost-devel`)*	—
Tutorials plotting	`matplotlib` 3.11	`python3.11-matplotlib` (or pip)	EPEL

System RPMs relied on (already installed on a stock Rocky 8, listed for completeness): `git`, `curl`, `tar`, `bzip2`, `coreutils`, `bash`, `glibc`, `glibc-devel` (for headers via the conda sysroot fallback), `environment-modules` (the `module` command was present though unused).

---

6. Everyday use

```
source ~/alps-env.sh          # every new shell (or add to ~/.bashrc)
spinmc --help
```

Job scripts (Slurm/PBS) must `source ~/alps-env.sh` too — a run that sees a different environment than the build will fail at load time on missing `.so`s.

Building ALPS 2.3.4 from source on macOS without root

Apple Silicon and Intel — micromamba or Homebrew

1 September 2026

Contents

Overview

macOS has no system package manager and its `/usr/local` (Intel) or `/opt/homebrew` (Apple Silicon) trees are owned by your user, so a "non-root" build is the normal case. Two self-contained routes:

- **Route A — micromamba (conda-forge).** One static binary, everything under `$HOME`, identical philosophy to the Linux rootless build. Recommended for reproducibility and for Apple Silicon.
- **Route B — Homebrew.** Uses the system (Apple) Clang plus `libomp`, or a Homebrew GCC. Lighter if you already run Homebrew.

Either way CMake downloads and compiles **Boost 1.87** itself. Never point ALPS at a Homebrew or conda Boost.

Prerequisite (both routes): the Xcode Command Line Tools, for headers and `git`. This prompts a GUI installer, no admin password:

```
xcode-select --install
```

macOS specifics that recur below: shared libraries are `.dylib`, not `.so`; the runtime search path variable is `DYLD_FALLBACK_LIBRARY_PATH`, not `LD_LIBRARY_PATH`; do not mix `arm64` and `x86_64` (Rosetta) toolchains in one build.

Route A — micromamba

A.1 Install micromamba

```
mkdir -p "$HOME/bin" && cd "$HOME/bin"
# Apple Silicon:
curl -Ls https://micro.mamba.pm/api/micromamba/osx-arm64/latest | tar -xvj bin/micromamba
# Intel:
# curl -Ls https://micro.mamba.pm/api/micromamba/osx-64/latest  | tar -xvj bin/micromamba
mv bin/micromamba . && rmdir bin 2>/dev/null
./micromamba --version
```

If `micro.mamba.pm` misbehaves, use the release mirror, substituting `osx-arm64` or `osx-64`:

`https://github.com/mamba-org/micromamba-releases/releases/latest/download/micromamba-osx-arm64`

A.2 Create the environment

conda-forge uses the **Clang** toolchain on macOS (there is no `gcc_osx-*` metapackage the way there is on Linux):

```
export MAMBA_ROOT_PREFIX=$HOME/micromamba
ARCH=osx-arm64          # or osx-64

$HOME/bin/micromamba create -y -n alps -c conda-forge \
    clang_$ARCH clangxx_$ARCH gfortran_$ARCH \
    cmake make \
    hdf5 openblas \
    openmpi \
    python=3.11 numpy scipy
```

Pin CMake below 4 (ALPS's Boost super-build needs pre-4 policy support):

```
$HOME/bin/micromamba install -y -n alps -c conda-forge 'cmake>=3.24,<4'
```

A.3 Environment script

```
# ~/alps-env.sh  --  source ~/alps-env.sh
export MAMBA_ROOT_PREFIX=$HOME/micromamba
eval "$("$HOME/bin/micromamba" shell hook -s bash)"
micromamba activate alps

ARCH_TRIPLE=arm64-apple-darwin20.0.0        # x86_64-apple-darwin13.4.0 on Intel
export ALPS_PREFIX=$HOME/opt/alps
export CC=$(which ${ARCH_TRIPLE}-clang)
export CXX=$(which ${ARCH_TRIPLE}-clang++)
export FC=$(which ${ARCH_TRIPLE}-gfortran)

if [ -f "$ALPS_PREFIX/bin/alpsvars.sh" ]; then
    source "$ALPS_PREFIX/bin/alpsvars.sh"
    export PYTHONPATH="$ALPS_PREFIX/lib/python3.11/site-packages:$PYTHONPATH"
    export
     ↪  DYLD_FALLBACK_LIBRARY_PATH="$ALPS_PREFIX/lib:$CONDA_PREFIX/lib:$DYLD_FALLBACK_LIBRARY_PATH"
fi
```

(The exact `${ARCH_TRIPLE}` prefix is whatever `ls $CONDA_PREFIX/bin/*clang` shows.)

A.4 Configure, build, install

```
source ~/alps-env.sh
cd "$HOME/src" && git clone --depth 1 https://github.com/alpsim/ALPS alps-src

cmake -S alps-src -B alps-build \
      -DCMAKE_BUILD_TYPE=Release \
      -DCMAKE_INSTALL_PREFIX="$ALPS_PREFIX" \
      -DCMAKE_PREFIX_PATH="$CONDA_PREFIX" \
      -DPython3_EXECUTABLE="$(python -c 'import sys;print(sys.executable)')" \
      -DCMAKE_CXX_FLAGS="-DBOOST_NO_AUTO_PTR -DBOOST_FILESYSTEM_NO_CXX20_ATOMIC_REF"

cmake --build   alps-build -j"$(sysctl -n hw.perflevel0.logicalcpu 2>/dev/null || sysctl -n
 ↪  hw.ncpu)"
cmake --build   alps-build -t test
cmake --install alps-build
```

Route B — Homebrew

B.1 Packages

```
brew update
brew install cmake hdf5 open-mpi openblas libomp git \
             python@3.11 numpy scipy
```

`brew` writes to `/opt/homebrew` (Apple Silicon) or `/usr/local` (Intel), both user-owned — no `sudo`.

Apple's Clang has **no OpenMP** out of the box; `libomp` supplies it. If you would rather use a real GCC:

```
brew install gcc            # provides gcc-14 / g++-14 / gfortran-14
```

Do **not** `brew install boost`.

B.2 Configure with Apple Clang + libomp

```
export ALPS_PREFIX=$HOME/opt/alps
BREW=$(brew --prefix)
PY="$BREW/opt/python@3.11/bin/python3.11"
"$PY" -m pip install --user numpy scipy      # if the brewed numpy/scipy are not importable

cmake -S alps-src -B alps-build \
      -DCMAKE_BUILD_TYPE=Release \
      -DCMAKE_INSTALL_PREFIX="$ALPS_PREFIX" \
      -DCMAKE_PREFIX_PATH="$BREW;$BREW/opt/openblas;$BREW/opt/hdf5;$BREW/opt/libomp" \
      -DOpenMP_ROOT="$BREW/opt/libomp" \
      -DHDF5_ROOT="$BREW/opt/hdf5" \
      -DPython3_EXECUTABLE="$PY" \
      -DCMAKE_CXX_FLAGS="-DBOOST_NO_AUTO_PTR -DBOOST_FILESYSTEM_NO_CXX20_ATOMIC_REF"
```

With Homebrew GCC instead, add `-DCMAKE_C_COMPILER=$BREW/bin/gcc-14 -DCMAKE_CXX_COMPILER=$BREW/bin/g++-14 -DCMAKE_Fortran_COMPILER=$BREW/bin/gfortran-14` and drop `-DOpenMP_ROOT`.

B.3 Build / install — same as A.4

Runtime environment for Route B:

```
source "$HOME/opt/alps/bin/alpsvars.sh"
export PYTHONPATH="$HOME/opt/alps/lib/python3.11/site-packages:$PYTHONPATH"
export DYLD_FALLBACK_LIBRARY_PATH="$HOME/opt/alps/lib:$(brew
 ↪  --prefix)/lib:$DYLD_FALLBACK_LIBRARY_PATH"
```

Package reference

Dependency	conda-forge (Route A)	Homebrew (Route B)
C / C++ compiler	`clang_osx-arm64`, `clangxx_osx-arm64`	Apple Clang (CLT) + `libomp`, or `gcc`
Fortran compiler	`gfortran_osx-arm64`	`gcc` (gfortran-14)
OpenMP runtime	bundled with conda clang	`libomp`
CMake (>= 3.18, < 4)	`cmake`	`cmake`
Make	`make`	`make` (or Xcode)
HDF5	`hdf5`	`hdf5`
BLAS / LAPACK	`openblas`	`openblas`
MPI	`openmpi`	`open-mpi`
Python 3.11	`python=3.11`	`python@3.11`
NumPy / SciPy	`numpy`, `scipy`	`numpy`, `scipy`
Git	(system CLT)	`git`
Boost 1.87	built by CMake — do **not** add a package	same
(optional) plotting	`matplotlib`	`matplotlib` (pip)

Verification

```
source ~/alps-env.sh                    # Route A  (Route B: source the snippet in B.3)
which spinmc                            # ~/opt/alps/bin/spinmc
otool -L "$(which spinmc)" | head       # dylib deps resolve into ~/opt/alps or the env
python -c "import pyalps, numpy, scipy; print('ok')"
```

Quick physics check (2-D Ising, cluster updates):

```
import pyalps
p=[{'LATTICE':'square lattice','L':16,'MODEL':'Ising','J':1,'T':T,
    'THERMALIZATION':10000,'SWEEPS':50000,'UPDATE':'cluster'} for T in (2.0,2.27,2.8)]
inp=pyalps.writeInputFiles('ising',p)
pyalps.runApplication('spinmc',inp,Tmin=2,writexml=True)
for s in pyalps.flatten(pyalps.loadMeasurements(
        pyalps.getResultFiles(prefix='ising'), '|Magnetization|')):
    print(s.props['T'], s.y[0])
# |M| ~ 0.91 (T=2.0)  ->  ~0.7 (T=2.27)  ->  ~0.2 (T=2.8)
```

---

macOS troubleshooting

`ld: library not found for -lomp` (Homebrew + Apple Clang) — pass `-DOpenMP_ROOT="$(brew --prefix libomp)"`, wipe `alps-build`, reconfigure.

`dyld: Library not loaded: @rpath/libhdf5...` at runtime — add the env/brew `lib` directory to `DYLD_FALLBACK_LIBRARY_PATH` (see the env scripts). SIP does not strip `DYLD_*` from binaries you built yourself.

Configure picks `/usr/bin/python3` — always pass an explicit `-DPython3_EXECUTABLE=...`; the interpreter CMake selects is the one Boost.Python binds to and it must be the one with numpy/scipy.

Mixed architecture errors (`building for macOS-x86_64 but ... arm64`) — your shell is under Rosetta or you mixed an `osx-64` micromamba with an `arm64` Homebrew. Pick one architecture for the whole toolchain.

`Compatibility with CMake < 3.5 will be removed` fatal — CMake 4. Use a `cmake` pinned `>=3.24,<4` (Route A already does).