Title: A Guide to Building the MCNP 6.3 Code from Source

Author(s): Josey, Colin James
Bolding, Simon R.

Intended for: 2021 MCNP User Symposium, 2021-07-12/2021-07-16 (Los Alamos, New Mexico, United States)

Issued: 2021-07-06

A Guide to Building the MCNP® 6.3 Code from Source

Colin Josey, Simon Bolding

XCP-3 (Monte Carlo Codes)

2021 MCNP® User Symposium
July 13, 2021

Introduction

Since version 6.2, there have been a number of key changes to how the code is built:

- Compiler support has changed
- The CMake build system is now the only build system
- HDF5 is a required dependency
- Parallel HDF5 is an optional dependency for a few MPI-specific features

This presentation will briefly discuss what you need to know, and show how to set this up on conventional machines.

Note: All this information is up to date as of date of publication, but can and likely will change with time. For dependencies, it is recommended to check their building instructions for possible future changes.

Latest information will also appear in the README file in the MCNP source.

Compiler Support

Due to a number of infrastructure changes, MCNP now requires Fortran 2018 and C++14 support in the compiler.

Known working compilers:

- Intel 19, 19.1 compiler sets
  - The MCNP test suites will 100% pass with these compilers in single-threaded mode.
  - Intel 19.1 on macOS has a known issue with OpenMP[1]
  - Windows: only Intel Fortran is needed. Visual Studio provides C/C++
  - Linux: must use newer GCC standard library (see Slide 12)
  - macOS: Apple-Clang (XCode) is not supported as a C/C++ compiler

- GCC 7.4 or newer
  - Most but not all tests will pass with GCC due to different roundoff
  - GCC 11.1 has known bug that causes memory errors - fixed in 11.2.

[1] This is fixed in Intel OneAPI 2021.3, but MCNP is not yet rigorously tested against this compiler.

Required Build Dependencies: CMake, Git

CMake:

- CMake 3.16 or newer is required to build MCNP.
- Latest release can be downloaded here: https://cmake.org/download/ or from your favorite package manager.
- CMake just needs to be in your path at this point.

Git:

- Our dependency manager uses git to check versions and add information to the `mcnp6 -v` command
- Latest release can be downloaded here: https://git-scm.com/downloads or from your favorite package manager.
- Any version should work (1.8.3 earliest tested)

Required Build Dependencies: HDF5

HDF5 manages file IO for many components of MCNP (runtape, XDMF output, new UM capabilities).

- HDF5 1.10.2 or newer is required to build MCNP[2].
- We request the v1.10 file format to maximize compatibility regardless of HDF5 version.
- Latest release can be downloaded here https://portal.hdfgroup.org/display/support/Download+HDF5: or from your favorite package manager.
- See Slide 14 for detailed building guides for each OS.

[2] Future versions of HDF5 may require `-DH5_USE_112_API` added to the C flags if the API changes.

Optional Build Dependencies: MPI

MPI is only required to enable MPI parallelism.

- The implementation should support MPI-3 or newer
- Performance is generally[3] insensitive to the MPI implementation
- If your cluster has an already installed MPI, that should probably be used

[3] Except FMESH Batch RMA, which tends to be fastest with OpenMPI.

Optional Build Dependencies: Parallel HDF5

MCNP, even with MPI, does not require parallel HDF5. However, two MPI-only features require parallel HDF5:

- MPI-Parallel PTRAC (threading available without parallel HDF5)
- FMESH parallel HDF5+XDMF results output

Parallel HDF5 must be built from source. Further, it must be built with the same MPI and compiler that will build MCNP. **Mismatches will cause build errors.**

Important:
Some parallel HDF5 features do not work on all file systems. In particular, NFS can cause parallel HDF5 IO to lock up.

Test your file system before running mission critical simulations.

Optional Build/Test Dependencies: X11, Python+h5py

The MCNP plotter relies on X11 to render graphics.

- On Linux, libraries should already be present
- On macOS, an X11 server such as XQuartz should be installed
- On Windows, an X11 library is included to allow building without it

Python 3 and h5py are only required when running tests. Both can be acquired from package managers or any scientific Python package. We do not support Python 2.

Configuring MCNP

The CMake configure syntax takes the form of:

```
cmake <path to MCNP source root> [-D OPTIONS=VAL]
```

Useful options are:

Option	Default	Effect
`MPI`	OFF	Enable or disable MPI
`OpenMP`	ON	Enable or disable OpenMP threading
`mcnp.plotter`	ON	Enable or disable plotter
`CMAKE_INSTALL_PREFIX`	OS-dependent	Installation path for MCNP

An example input line that has MPI enabled, the plotter disabled, and is built in a folder called "build" within the MCNP source tree:

```
cmake .. -D MPI=ON -D mcnp.plotter=OFF
```

There are many more options. See the MCNP 6.3 build guide for more detail.

Building/Testing/Installing MCNP

The three commands to build, test, and install MCNP are:

```
cmake --build . [--config Release]
ctest [--build-config Release]
cmake --install . [--config Release]
```

On Visual Studio, append the portion in brackets to all 3 for optimization. Note that not all tests are expected to pass unless you are using the Intel 19.0 compiler and are not using MPI.

The `cmake` and `ctest` commands can be run in parallel by adding `-j N` (where `N` is the number of processes) to the end of the command.

If tests are failing, `ctest --output-on-failure` can give more information.

Los Alamos NATIONAL LABORATORY

Extra Notes: Windows

To build MCNP natively you will need:

- Visual Studio 2019 or newer (https://visualstudio.microsoft.com/vs/)
- “Desktop development with C++” module installed
- Intel Fortran installed on top
- CMake installed (with the path added for the user)
- (Optional) MS-MPI (https://docs.microsoft.com/en-us/message-passing-interface/microsoft-mpi, get both setup and SDK)

You can also install MCNP inside of Windows Subsystem for Linux (WSL). For performance reasons, WSL 2 is highly recommended over WSL 1. If this route is taken, follow the Linux instructions instead.

Extra Notes: Linux

If building with the Intel compiler, it borrows GCC's standard libraries. This presents a problem as, while Intel 19 itself will build MCNP, the GCC library may be too out of date to be used.

MCNP needs a relatively recent standard library. If you get build errors for "`enable_if_t`", tell Intel to use a newer library. We test with GCC 7.4.

On Intel 19 / 19.1:

```
export CXXFLAGS="-gxx-name=<path to g++ from target GCC>"
```

On Intel OneAPI:

```
export CXXFLAGS="--gcc-toolchain=<path to g++ from target GCC>"
```

Extra Notes: Spack

Spack (https://spack.readthedocs.io/en/latest/) is a build and package manager for Linux and macOS. It can greatly simplify the installation process, once set up correctly.

If Spack is installed and the correct compiler is set as the default, the CMake, MPI, and parallel HDF5 build process can be compacted to:

```
spack install cmake
spack install openmpi
spack install hdf5+mpi^openmpi
```

Then, load all three prior to building MCNP (either using `spack load` or `module load`, depending on how you set up Spack).

Building HDF5

The following slides will show how to build HDF5 on Windows, Linux, and macOS. There are a few notes:

- Historically, the build process has changed between HDF5 versions.
- All of these are tested on 1.12.0 and 1.10.7 in both non-parallel and parallel configurations.
- In general, the CMake-built HDF5 is preferred due to stability, but on macOS, the CMake script is hardcoded to build with Apple-Clang, which poses problems.

Building HDF5, Windows

Get `CMake-hdf5-<version>.zip` from the HDF5 website and extract it. Then, from a CMD terminal with CMake and Visual Studio set up:

```
cd CMake-hdf5-<version>
ctest -S HDF5config.cmake,BUILD_GENERATOR=VS201964 -C Release -V -O hdf5.log
```

Adjust the build generator depending on which version of Visual Studio you have installed.
Then, extract the generated `HDF5-<version>-win64.zip` file to a location of your choice, and set the environment variable `HDF5_DIR` to point to:

```
set HDF5_DIR=<install path>\cmake\hdf5
```

For parallel HDF5, replace the `ctest` line with:

```
ctest -S HDF5config.cmake,BUILD_GENERATOR=VS201964 -C Release -V -O hdf5.log -D
    MPI=ON -D LOCAL_SKIP_TEST=ON
```

Tests are disabled, as the parallel tests tend to lock up on NTFS. If you have a true parallel file system, that can be omitted.

Building HDF5, Linux

Get `CMake-hdf5-<version>.tar.gz` from the HDF5 website. Then:

```
tar xf CMake-hdf5-<version>.tar.gz
cd CMake-hdf5-<version>
./build-unix.sh

# The below commands can be performed wherever convenient.
tar xf HDF5-<version>-Linux.tar.gz
export HDF5_DIR="$(pwd)/HDF5-<version>-Linux/HDF_Group/HDF5/<version>
    /share/cmake/hdf5"
```

For parallel HDF5 1.12.0+, replace the `./build-unix.sh` line with:

```
ctest -S HDF5config.cmake,BUILD_GENERATOR=Unix -C Release -V -O hdf5.log -D MPI=
    ON
```

For parallel HDF5 1.10.7:

```
./build-unix-hpc.sh
```

Building HDF5, macOS

Get `hdf5-<version>.tar.gz` from the HDF5 website. Then:

```
export HDF5_ROOT=<install path>

tar xf hdf5-<version>.tar.gz
cd hdf5-<version>
./configure --prefix=$HDF5_ROOT
make -j && make test && make install
```

For parallel HDF5, replace the `configure` line with:

```
CC=mpicc ./configure --enable-parallel --prefix=$HDF5_ROOT
```