LA-UR-25-22869

Approved for public release; distribution is unlimited.

Title: MCNP® Code Version 6.3.1 Build Guide

Author(s): Bull, Jeffrey S.
Josey, Colin James
Kulesza, Joel A.
Rising, Michael Evan
Swaminarayan, Sriram

Intended for: Report

Issued: 2025-04-02 (rev.1)

Los Alamos
NATIONAL LABORATORY

Los Alamos National Laboratory, an affirmative action/equal opportunity employer, is operated by Triad National Security, LLC for the National Nuclear Security Administration of U.S. Department of Energy under contract 89233218CNA000001. By approving this article, the publisher recognizes that the U.S. Government retains nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes. Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy. Los Alamos National Laboratory strongly supports academic freedom and a researcher's right to publish; as an institution, however, the Laboratory does not endorse the viewpoint of a publication or guarantee its technical correctness.

LA-UR-25-22869, Rev. 1

MCNP®

Code Version 6.3.1

Build Guide

Document compiled from `git` hash `84851ff318` on April 1, 2025.

This document cannot be made Section-508 compliant (accessible to persons with disabilities) at this time. If you need help accessing this document, please email mcnp_help@lanl.gov.

Los Alamos
NATIONAL LABORATORY

MCNP® Code Version 6.3.1 Build Guide

LA-UR-25-22869, Rev. 1

April 1, 2025

Los Alamos National Laboratory

Jeffrey S. Bull
Colin Josey

Joel A. Kulesza
Michael E. Rising

Sriram Swaminarayan

MCNP® and Monte Carlo N-Particle® are registered trademarks owned by Triad National Security, LLC, manager and operator of Los Alamos National Laboratory. Any third party use of such registered marks should be properly attributed to Triad National Security, LLC, including the use of the ® designation as appropriate. Any questions regarding licensing, proper use, and/or proper attribution of Triad National Security, LLC marks should be directed to trademarks@lanl.gov.

Disclaimer: Los Alamos National Laboratory, an affirmative action/equal opportunity employer, is operated by Triad National Security, LLC for the National Nuclear Security Administration of U.S. Department of Energy under contract 89233218CNA000001. By approving this article, the publisher recognizes that the U.S. Government retains nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes. Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy. Los Alamos National Laboratory strongly supports academic freedom and a researcher's right to publish; as an institution, however, the Laboratory does not endorse the viewpoint of a publication or guarantee its technical correctness.

Contents

1 Introduction

This is a build guide for the MCNP® code, version 6.3.1, that expands upon the `README.md` included with the source code. It covers compilers, dependencies, building, testing, and installing the code in one of its supported configurations.

Caution

These instructions are focused on building the MCNP6.3.1 code with limited information on obtaining and/or building the required and/or optional software the code depends on. The reader is urged to review the latest details on building the code at the Build Guide Details page on the MCNP website where more details on the peripheral software and nuances of each supported operating system can be found.

2 Compilers

The MCNP code can be built with any compiler set that supports the Fortran 2018, C99, and C++14 compiler standards. However, as of the release of this document, only two compiler sets (Intel and GNU) that meet these requirements have been tested by the MCNP Development Team; these are listed in Table 1.

Table 1: Compilers Sets Known to Successfully Build MCNP6.3.1

Compiler Set	OS	Minimum Working Version Tested	Latest Version Tested
Intel oneAPI C/C++ compilers (`icx`/`icpx`) with Intel Classic Fortran Compiler (`ifort`)	Linux, Windows	2024.2	* 2024.2
** Intel Classic Compilers (`ifort`/`icc`/`icpc` on macOS)	macOS (x86_64)	19.0	* Intel oneAPI 2021.5 on macOS (x86_64)
** Intel Classic Compilers (`ifort`/`icc`/`icpc` on Linux and `ifort`/`icl` on Windows)	Linux, Windows	19.0	icc 2021.10, released with Intel oneAPI 2023.2 ifort 2021.13.0, released with Intel oneAPI 2024.2
GCC	Linux, macOS (x86_64 and arm64), Windows Subsystem for Linux (WSL)	9.4	14.2.0

* The distributed MCNP6.3.1 binaries are built using a configuration of Intel oneAPI and Classic compilers on Linux and Windows, and using the Intel Classic Compilers on macOS (x86_64). Specific versions of the compilers used are given in the release notes.
** The Intel Classic Compilers are no longer distributed in the latest Intel oneAPI releases.

The MCNP Development Team recommends using the Intel compilers to build the MCNP code. The options available with these compilers allow the compiled MCNP executables to provide consistent results across all supported systems. The MCNP6.3.1 distributed executables are built using the Intel compilers.

⚠ Caution

At the time of the release of the MCNP6.3.1 code, a combination of the Intel oneAPI compilers (`icx`/`icpx`) and the Classic Fortran compiler (`ifort`) is preferred to build the code. While the Intel oneAPI Fortran compiler (`ifx`) may be able to build the code on certain OS's, this compiler is not yet feature complete, does not provide consistent results across OS's, and is therefore not considered an established compiler that can be used to build production binaries of the code.

Links to the supported compilers and additional MCNP Development Team recommendations can be found on the Build Guide Details page on the MCNP website.

3 Dependencies

The MCNP source code has several software dependencies which must be installed to build and test the MCNP executables. These can be broken down into three categories: dependencies needed to compile the code (§3.1), test the code (§3.2), and other optional dependencies (§3.3).

3.1 Dependencies Required To Compile the MCNP Code

The primary dependencies needed to compile the MCNP6.3.1 code are listed in Table 2.

Table 2: Dependencies Required To Compile the MCNP Code

Component	Minimum Version	Latest Version Tested
CMake	3.20	4.0.0
Git	Any	2.48.1
HDF5	1.10.2 on Linux	2.0.0-devel on Linux
	1.14.3 on macOS	1.14.6 on macOS
	1.10.2 on Windows	1.14.6 on Windows

In addition to the specific software required to build the code, some general build tools must also be installed. For Linux/macOS, Unix Makefiles (e.g., GNU Make) are recommended. For Windows, Build Tools for Visual Studio are recommended. To make use of these build instructions, Unix Makefiles or Build Tools for Visual Studio are required on Linux/macOS or Windows, respectively. For more operating system-specific build tool recommendations, see the MCNP Development Team build guidance for Linux, macOS, and Windows platforms on the MCNP website.

3.1.1 CMake

The MCNP software package uses the CMake build system generator to configure, compile, test, install, and package the MCNP code. With CMake, a variety of build tools, such as GNU Make, Ninja, MSBuild, and many others can be configured to compile the source code. Which build tool CMake uses is determined by the CMake generator. The default CMake generator for Linux and macOS is Unix Makefiles (GNU Make) and for Windows it's the latest version of Visual Studio. These defaults can be changed using the CMake `-G` option.

There are several ways CMake can be obtained for each operating system. For detailed information for each operating system, see the MCNP Development Team build guidance for Linux, macOS, and Windows platforms on the MCNP website.

3.1.2 Git

Git is required within the build system to check for consistency of the internal dependencies. There are several ways Git can be obtained for each operating system. For detailed information for each operating system, see the MCNP Development Team build guidance for Linux, macOS, and Windows platforms on the MCNP website.

3.1.3 HDF5

HDF5 provides the binary file I/O system for the MCNP code as of version 6.3.1. It is available in sequential and parallel (MPI) versions. Uses of the parallel version are discussed in §3.3.1.

There are several ways HDF5 can be installed or built on each operating system. For detailed information for each operating system, see the MCNP Development Team build guidance for Linux, macOS, and Windows platforms on the MCNP website. To configure the code on macOS, the `-DHDF5_USE_STATIC_LIBRARIES=OFF` variable needs to be set. For Windows, suitable versions with a `hdf5-<version-number>-win-vs2022_intel.msi` installer package are typically the easiest to install.

Due to changes made to the HDF5 build system from version to version, it may be beneficial to use version 1.12.1 of the HDF5 library on Windows and/or Linux. Some helpful instructions on building this library for the previous MCNP6.3.0 release can be found here.

3.2 Dependencies for Testing

The MCNP test system uses Python3 to run the tests and compare the results to a set of test templates. Testing is recommended to ensure the MCNP code is built correctly, but these tests are only expected to result in 100% passing with the Intel compilers. The minimum version of Python3 and necessary modules are listed in Table 3.

Table 3: Dependencies Required for Testing

Component	Minimum Version
Python3	3.9
NumPy	Any
h5py	Any

There are several ways Python3 can be obtained for each operating system. For detailed information for each operating system, see the MCNP Development Team build guidance for Linux, macOS, and Windows platforms on the MCNP website.

3.3 Optional Dependencies

These dependencies are only required for specific features.

3.3.1 MPI

To run the MCNP code across multiple machines or nodes on a cluster (or with certain model physics in parallel), MPI-3 is required.

There are a wide variety of MPI providers and most should work. The MCNP Development Team actively tests OpenMPI and MPICH on both Linux and macOS, MVAPICH and Intel-MPI on Linux, and MS-MPI on Windows.[1] If using OpenMPI with parallel HDF5, there is an MPI-IO data corruption bug that was fixed in 3.1.4 for the 3.x series and 4.0.1 for the 4.x series. It is highly recommended to use these versions or newer.

In general, the MCNP code's performance is insensitive to the MPI implementation used. The remote memory access (RMA) feature used in the Batch RMA mode in `FMESH` is one exception. In the MCNP team's testing, OpenMPI was found to be much faster than MPICH and its derivatives for this specific feature. On a simple 1 million tally Batch RMA problem, the code built with OpenMPI 4.0.4 ran 10.5 times faster than MPICH 3.3.2.

There are several ways MPI providers can be obtained for each operating system. For detailed information for each operating system, including tips on obtaining and/or building MPI-enabled HDF5, see the MCNP Development Team build guidance for Linux, macOS, and Windows platforms on the MCNP website.

Note that an MPI version of the MCNP executables can be compiled with either the serial or parallel version of HDF5. However, building an MPI version of the MCNP code with a sequential version of HDF5 will disable these features:

- MPI-parallel PTRAC (OpenMP-parallel PTRAC using `tasks N` will still work with or without parallel HDF5).
- `FMESH` parallel file writing via the `PIO` card.

If these features are desired, a parallel build of HDF5 is required.

3.3.2 X11 Development Libraries

X11 is used by the MCNP plotting package. By default, the MCNP code is configured to build with the plotter enabled and will not compile if the X11 development files are not available. To build without requiring X11, add `-Dmcnp.plotter=OFF` to the CMake configuration command line (§4.1.1).

There are several ways X11 can be obtained for each operating system. For detailed information for each operating system, see the MCNP Development Team build guidance for Linux, macOS, and Windows platforms on the MCNP website.

3.3.3 The `chrpath` Package (Linux only)

This optional dependency is only needed when packaging the code on Linux. See §4.1.5 for more information on the packaging process. For more information on `chrpath`, see the MCNP Development Team build guidance for Linux.

[1]Although MS-MPI is only MPI-2 compliant, it has all the necessary MPI-3 features used by the MCNP code.

4 Building the Code

The building process for the MCNP code has five major steps: configuring, compiling, testing, installing, and packing for distribution. This section provides build steps in §4.1 that are generally independent of the operating system being used. Additionally, a section on build steps specifically for Linux / macOS is provided in §4.2, and a section on build steps specifically for Windows is provided in §4.3. Because of the complexities and evolving compiler support, primarily related to building on Apple silicon (arm64), some unsupported build workflows are provided on the MCNP6.3.1 release webpage as a courtesy to those needing help getting started.

4.1 General Build Steps

4.1.1 Configuring

The root `CMakeLists.txt` for the CMake build system is located in the `mcnp6` directory. Before running CMake, create a new directory/folder in the `mcnp6` directory named `build`.

```
cd <path to mcnp6>
mkdir build
cd build
```

During configuration, CMake will search the environment for the needed compilers and dependencies. If CMake cannot find or finds incorrect compilers, the compilers can be specified either on the CMake configuration command with the `CMAKE_C_COMPILER`, `CMAKE_CXX_COMPILER`, and `CMAKE_Fortran_COMPILER` variables or through the `CC`, `CXX`, and `FC` environment variables. For example, the compilers can be specified on the CMake command line:

```
cmake -DCMAKE_C_COMPILER=icx -DCMAKE_CXX_COMPILER=icpx -DCMAKE_Fortran_COMPILER=ifort ..
```

The Linux / macOS (§4.2) and Windows (§4.3) example build sections assume the user environment is configured with the expected compilers in place; therefore, the CMake compiler variables are generally not shown even though they may be needed.

Caution

As mentioned in §2, only the Intel Classic Fortran compiler (`ifort`) is used for the production builds of the MCNP6.3.1 code. Therefore, it is likely that the CMake selected Fortran compiler will need to be overridden using either the `CMAKE_Fortran_COMPILER` CMake variable or the `FC` environment variable when using the current Intel compilers. Note the Intel Classic Fortran compiler (`ifort`) was last distributed in version 2024.2 of the Intel compilers.

See §A.2.1 for more information on all of the MCNP-specific CMake variables that are most relevant to building the code.

4.1.1.1 Default Configuration The default configuration builds the MCNP code in `Release` mode (see `CMAKE_BUILD_TYPE` in §A.1) with both OpenMP threading and the MCNP plotter enabled. This configuration was used to build the distributed non-MPI MCNP6.3.1 production executables.

```
cmake ..
```

4.1.1.2 MPI Configuration

The MPI configuration can be as simple as executing:

```
cmake -DMPI=ON ..
```

However, additional MPI configuration options may be needed depending on if and how the parallel HDF5 was installed. Some discussion on additional CMake configuration variables that may be needed for Linux / macOS and Windows operating systems can be found in §4.2.1 and §4.3.1, respectively.

Note the MCNP code can be built in an MPI configuration regardless if using a serial or parallel HDF5 library. See §3.3.1 for further details.

4.1.2 Compiling

Once the MCNP code has been configured, it can be compiled by running the selected build tool or by using the CMake command:

```
cmake --build . [--config Release] -j N
```

The `--config Release` option is required if the build tool specified by the CMake generator creates all the `CMAKE_BUILD_TYPE` configurations at once. For example, the Visual Studio generators create configurations for each of the standard `CMAKE_BUILD_TYPE` options `Release`, `Debug`, `RelWithDebInfo`, and `MinSizeRel`. This option is not needed for the Unix Makefile or Ninja build tools.

The `-j N` option allows the build tool to compile the code in parallel, with `N` being the number of processing cores used to concurrently compile the code.

4.1.3 Testing

The MCNP software package uses the CTest tool included with CMake to run the test sets. For more information about CTest, see the CTest documentation. See §A.2.2 for more information on all of the MCNP-specific CMake variables that are most relevant to testing the code.

The tests are a series of primarily integral tests that compare the solution to a single-threaded Intel oneAPI 2023.2-compiled (using the Intel oneAPI C/C++ (`icx`/`icpx`) and Intel Classic Fortran (`ifort`) compilers) Linux version of the MCNP code. Due to the characteristics of the Monte Carlo method (i.e., a single branch change from floating-point round-off differences may induce cascading changes), it is likely that the MCNP code compiled in any other way will have several differences when compared against these templates. Any differences should be evaluated to determine the acceptability of the resulting executable.

To assist in assessing the test result differences, the CMake option `mcnp.test.tolerance` can be used to set the percent tolerance when comparing the tally results with the templates. By default, the value of `mcnp.test.tolerance` is `0.0`.

Note: all the additional CMake options listed in §A.2.2 can be included in the original CMake configuration step, which avoids the need to rerun CMake just to configure the tests.

Testing using one thread (tasks 1) By default, the MCNP test system is configured to run the tests with only one thread. To run the tests, enter:

```
ctest -j N
```

where `N` is the number of tests being executed concurrently.

Testing more than one thread (tasks n) To run the MCNP tests using more than one thread, CMake must be rerun to configure the test system. Use this CMake command to configure the build system to run the tests using threading.

```
cmake . -Dmcnp.test.nomp=n -Dmcnp.test.tolerance=0.01
```

where `n` is the number of threads per test to use when running the test suite.

Since there are differences in the `outp` file when the MCNP code is run with threads, CTest should be told to ignore the output files results. To do this, use this command to run the tests:

```
ctest -j N -E outp
```

Testing MPI builds When the build system is configured to build the MPI version of the MCNP executables, by default the test system is configured to use 3 MPI processes during testing. This value can be changed by using the CMake `mcnp.test.nmpi` option. As when testing using more than one thread, the option `mcnp.test.tolerance` should be set. The recommended CMake configuration command is:

```
cmake . -Dmcnp.test.nmpi=n -Dmcnp.test.tolerance=0.01
```

where `n` is the number of MPI processes to use when running the test suite. Once the build system is configured, use this command to run the tests:

```
ctest -j N -E "outp|ptrc|produces.*[0-9]h"
```

If using a serial HDF5 library, then four tests associated with the `mcnp.regression.inp02` are expected to fail when executing the tests with the MCNP MPI executable.

4.1.4 Installing

To install the MCNP software package, run the install option for the selected build tool or use the CMake command:

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

In all cases, the MCNP executable will be located in the directory/folder `CMAKE_INSTALL_PREFIX/bin`.

The `--config Release` option is required if the build tool specified by the CMake generator creates all the `CMAKE_BUILD_TYPE` configurations at once. For example, the Visual Studio generators create configurations for each of the standard `CMAKE_BUILD_TYPE` options `Release`, `Debug`, `RelWithDebInfo`, and `MinSizeRel`. This option is not needed for the Unix Makefile or Ninja build tools recommended for Linux / macOS and Windows, respectively.

Caution

The MCNP executables depend on several compiler-specific dynamic libraries which will be needed when executing the version installed using the `cmake --install .` method described herein. Therefore, any user that runs the MCNP executable installed in this fashion must also have access to the compiler and its associated libraries. For computer systems that use modules to allow users access to several different compilers, it is recommended that the MCNP software be installed from a distribution package instead of using the install option. See §4.1.5.

4.1.5 Packing for Distribution

The CMake build system also includes the CPack tool, which allows one to pack up the code executable, associated libraries, and source code into a compressed archive and install it on a different computer. This is the preferred method for installing the MCNP software package on computer networks in which several users need access to the MCNP executables. For more information about CPack, see the CPack documentation. See §A.2.3 for more information on all of the MCNP-specific CMake variables that are most relevant to packaging the code.

To configure the MCNP software package for a self-contained distribution, use this command:

```
cmake . -Dmcnp.install.bundle.dependencies=ON
```

To include the source code in the distribution package, add the `mcnp.install.bundle.source` option set to `ON` to the CMake command.

```
cmake . -Dmcnp.install.bundle.source=ON
```

To create the MCNP distribution package, enter:

```
cpack
```

A resulting archive file of the built executables and any additional bundled dependencies and/or source code is created and can be moved to the machine where it may be unpacked and installed.

4.2 Building on Linux / macOS with Make

⚠ Caution

The macOS instructions that follow are for building the MCNP6.3.1 code on x86_64 architecture Apple machines. The Intel Classic compilers needed to build on this architecture were last distributed in the Intel oneAPI 2024.0 version. Because of the complexities and evolving compiler support, primarily related to building on Apple silicon (arm64), some unsupported build workflows are provided on the MCNP6.3.1 release webpage as a courtesy to those needing help getting started.

Assuming the environment is configured properly, with the expected compilers installed and all of the dependencies available, the following set of directions can be used to build, test, and install the MCNP code on a Linux / macOS system using the default Unix Makefile generator.

1. Open a terminal with a properly configured environment.
2. To create a build tree, execute:

```
cd <path to mcnp-6.3.1-Source>/mcnp6
mkdir build
cd build
```

3. To configure the code,
   (a) For the default configuration, execute:

   ```
   cmake ../
   ```

   (b) For the MPI configuration, execute:

   ```
   cmake -DMPI=ON ../
   ```

   For some configuration tips with respect to parallel HDF5, see §4.2.1.

   Add the `-DCMAKE_Fortran_COMPILER=ifort` variable to the `cmake` configuration commands above if the Intel Classic Fortran compiler is not properly found during configuration. If the Intel Classic C/C++ compilers are needed, adding the `-DCMAKE_C_COMPILER=icc -DCMAKE_CXX_COMPILER=icpc` variables to the `cmake` configuration commands may be needed to override the Intel oneAPI compilers.

   ⚠ Caution

   If configuring the MCNP code on Windows Subsystem for Linux AND a Windows version of HDF5 is installed, the Windows version of HDF must be removed from the PATH environment variable. Assuming HDF5 was installed in the default location (`C:\Program Files\HDF_Group\HDF5\1.xx.x`), enter this command in the WSL terminal window.

   ```
   export PATH=$(echo $PATH | tr ":" "\n" | grep -v "HDF_Group" | tr "\n" ":")
   ```

   This will remove the folder `HDF_Group` and all subfolders from the PATH environment variable.

4. To compile the code, execute:

```
make -j N
```

where N is the number of source files allowed to compile concurrently.

5. To test the code, execute:

```
ctest -j N
```

where N is the number of tests being executed concurrently.

6. To install the code, execute:

```
make install
```

where the MCNP executable will be located in the directory/folder `CMAKE_INSTALL_PREFIX/bin`.

7. To package the code, execute:

```
cpack -G TGZ
```

This command creates a `mcnp-6.3.1-production_YYYY_MM_DD_exec_<OS>.tar.gz` file.

4.2.1 Parallel HDF5

If both serial and parallel HDF5 were installed using a third-party package manager, CMake may need to be told to use the parallel version. Setting the `HDF5_PREFER_PARALLEL` CMake variable to `TRUE` during configuration may be needed to select the parallel over the serial version of HDF5:

```
cmake -DMPI=ON -DHDF5_PREFER_PARALLEL=TRUE ../
```

If parallel HDF5 was built from the source code using CMake, set the `HDF5_DIR` environment variable to the cmake directory inside of the HDF5 installation directory before running CMake to configure the MCNP code:

```
export HDF5_DIR=<path to parallel HDF5 installation folder>/share/cmake
cmake -DMPI=ON ../
```

If parallel HDF5 was built from the source code using GNU Autoconf (configure), set the `HDF5_ROOT` environment variable to the installation directory before running CMake to configure the MCNP code:

```
export HDF5_ROOT=<path to parallel HDF5 installation folder>
cmake -DMPI=ON ../
```

For detailed information on installing or building parallel HDF5 for Linux / macOS, see the MCNP Development Team build guidance for Linux and macOS platforms on the MCNP website.

4.3 Building on Windows with Ninja

Assuming the environment is configured properly, with the expected compilers installed and all of the dependencies available, the following set of directions can be used to build, test, and install the MCNP code on a Windows system using the Ninja generator. Note that the Ninja build tool is not the default generator for CMake, but it is the MCNP Development Team recommended build tool on Windows.

1. From the Start Menu, open Intel oneAPI 2022 → Intel Command Prompt for Intel 64 for Visual Studio 2022 and enter these commands:
2. To create a build tree, execute:

```
cd <path to mcnp-6.3.1-Source>\mcnp6
mkdir build
cd build
```

3. To configure the code,
   (a) For the default configuration, execute:

   ```
   set FC=ifort
   cmake -G Ninja ..\
   ```

   (b) For the MPI configuration, execute:

   ```
   set FC=ifort
   cmake -G Ninja -DMPI=ON -DMPI_GUESS_LIBRARY_NAME=MSMPI ..\
   ```

   The CMake `MPI_GUESS_LIBRARY_NAME` option shown in the above `cmake` configuration command must be set to `MSMPI` when both Microsoft MPI and Intel MPI are installed on the system.

   For some configuration tips with respect to parallel HDF5, see §4.3.1.

   Note the use of the `-G Ninja` option to specify that CMake generates build files for the Ninja build tool. Ninja is the build tool recommended for use with the Intel compilers on Windows systems.

   As an alternative to setting the `CC`, `CXX`, and `FC` environment variables, shown above in the command preceding the `cmake` command, the `CMAKE_<LANG>_COMPILER` variables can be added to the `cmake` configuration commands to specify the Intel Classic compilers.

4. To compile the code, execute:

```
ninja -j N
```

   where `N` is the number of source files allowed to compile concurrently.

5. To test the code, execute:

```
ctest -j N
```

   where `N` is the number of tests being executed concurrently.

6. To install the code, execute:

```
ninja install
```

where the MCNP executable will be located in the directory/folder `CMAKE_INSTALL_PREFIX\bin`.

7. To package the code, execute:

```
cpack -G ZIP
```

This command creates a `mcnp-6.3.1-production_YYYY_MM_DD_exec_<OS>.zip` file.

4.3.1 Parallel HDF5

Before running CMake, set the `HDF5_DIR` environment to the cmake folder in the parallel HDF5 installation folder. For HDF5 enter these commands:

```
set HDF5_DIR="<path to parallel HDF5 installation folder>\share\cmake"
cmake -G Ninja -DMPI=ON -DMPI_GUESS_LIBRARY_NAME=MSMPI ..\
```

For detailed information on installing or building parallel HDF5 on Windows, see the MCNP Development Team build guidance for Windows platforms on the MCNP website.

A Configuration Options

The following is a summary of useful standard CMake and MCNP-specific CMake variables that the MCNP build system provides. These variables can be used to change the MCNP build system configuration. To use them, add the variable name preceded by `-D` on the command line. Values not listed are intended for members of the MCNP Development Team.

A.1 Standard CMake Configuration Variables

`CMAKE_INSTALL_PREFIX` Default Value: OS-dependent

Tells CMake where to install the MCNP software package. Note that the default value of this variable will require administrative privileges to install the software. For more information, see the CMake documentation.

`CMAKE_BUILD_TYPE` Default Value: `Release`

Controls the performance optimization settings for the code. When set to `Release`, the MCNP code will build using the MCNP Development Team's recommended optimization settings. When set to `Debug`, optimization is disabled, and debug symbols are compiled in. For more information, see the CMake documentation.

`CMAKE_GENERATOR, -G` Default Value: OS-dependent

The generator used to build the project. For more information, see the CMake documentation.

`CMAKE_<LANG>_COMPILER` Default Value: None

Path to or name of compiler for `<LANG>`. Can be used to override CMake found C, C++, and Fortran compilers. For more information, see the CMake documentation.

`HDF5_PREFER_PARALLEL` Default Value: `False`

When both serial and parallel HDF5 are installed and found by CMake, this option allows the user to override the default to prefer the parallel version of HDF5 found. For more information, see the CMake documentation.

`MPI_GUESS_LIBRARY_NAME` Default Value: None

When multiple MPI implementations are installed and found, can specify a preferred MPI implementation to use. For more information, see the CMake documentation.

`BUILD_TESTING` Default Value: `ON`

Enables or disables the MCNP testing system. Setting this to `OFF` can be useful if one does not intend to test and the file system is particularly slow. For more information, see the CTest documentation.

A.2 MCNP-specific CMake Configuration Variables

A.2.1 Building Options

`mcnp.default_configuration` Default Value: `ON`

CMake has a set of "default" options it uses if otherwise not specified. Enabling this overrides these values and uses the MCNP Development Team's recommended options.

`OpenMP` Default Value: `ON`

Toggles building the MCNP code with OpenMP.

`MPI` Default Value: `OFF`

Toggles building the MCNP code with MPI. If set to `ON`, the MCNP code will attempt to find an MPI library and build an MPI-parallel version of the MCNP code. See §3.3.1.

`mcnp.plotter` Default Value: `ON`

Toggles building the MCNP plotter. If set to `OFF`, the MCNP code no longer depends on X11, and the plotter is disabled.

`mcnp.utilities` Default Value: `OFF`

Toggles building the utilities that come with the MCNP code, which make the code easier to use. Setting this to `ON` builds these utilities in addition to the MCNP code.

`shacl.<module>.shared_library` Default Value: `BUILD_SHARED_LIBS`

Determines if an internal dependency should be built as shared or not. Can be useful to adjust on systems where shared or static libraries are malfunctioning. Adjusting `BUILD_SHARED_LIBS` will control this value for most libraries.

A.2.2 Testing Options

`mcnp.test.nomp` Default Value: `1`

Number of OpenMP threads to use per test.

`mcnp.test.nmpi` Default Value: `3`

Number of MPI processes to use per test.

`mcnp.test.tolerance` Default Value: `0.0`

Percent tolerance for acceptance of floating-point differences.

`mcnp.test.useinstall` Default Value: `OFF`

Run the tests using the installed version of the MCNP software package. See §4.1.4.

`mcnp.unit_tests` Default Value: `ON`

Build and run unit tests for MCNP code. Set to `OFF` if `mcnp.test.useinstall` is `ON`.

`mcnp.test.sleep_after_run` Default Value: `0.0`

Time in seconds to sleep after running each MCNP test. Useful on systems with issues leaving files locked.

A.2.3 Packaging Options

`mcnp.install.bundle.dependencies` Default Value: `OFF`

Bundles all dependency libraries with the installation, typically installed in the `lib` directory under the `CMAKE_INSTALL_PREFIX` path.

`mcnp.install.bundle.source` Default Value: `OFF`

Bundles the full source code with the installation, typically installed in the `src` directory under the `CMAKE_INSTALL_PREFIX` path.

A.2.4 Advanced Options

These are advanced features that usually do not need to be changed. The following variables have specific uses that may be of value to investigate beyond the recommended default values. However, most of these variables correspond to the inner machinery of the build system.

`OPTIMIZATION_Release_<LANG>` Default Value: `-O2`

Controls the optimization level in release builds. This value will override any values in `CMAKE_<LANG>_FLAGS`.

`OpenMP_Preferred_Language` Default Value: `Detect`

Using certain sets of mixed compilers can cause OpenMP to fail to operate correctly. As both C++ and Fortran are using OpenMP, both need to link to the same library. The build system attempts to detect which language provides an OpenMP library that is compatible with both compilers. This may not always work, so this value toggles the language used. Some combinations of compilers do not work because of library incompatibility or missing OpenMP support.

`git.submodule.packages.cache` Default Value:
`<path to mcnp-6.3.1-Source>/mcnp6/../mcnp-dependencies`

This is a pointer to where all the submodule libraries are. In released versions of the MCNP code, this value is set to the `mcnp-dependencies` folder alongside the source tree.

`MPI_THREAD_MULTIPLE` Default Value: `OFF`

This option allows some features in the MCNP code to send messages from multiple OpenMP threads at the same time. This capability improves the performance of MPI in hybrid OpenMP-MPI simulations and is used in the Batch RMA tally mode in `FMESH`.

This feature is disabled by default for a reason. Specific combinations of MPI libraries and cluster interconnects (e.g., OpenMPI 4.0 and Omni-Path) do not support RMA with `MPI_THREAD_MULTIPLE` enabled. The easiest way to test this capability is to build the MCNP code with this option enabled and run the following input file across several compute nodes with multiple tasks. If it runs to completion without error messages from the MPI library, then it is safe to leave the option enabled.

Listing 1: mpi_thread_multiple_test.mcnp.inp.txt

```
MPI_THREAD_MULTIPLE test script
1        0                          -1       imp:n=1
2        0                           1       imp:n=0

1        so       500

```

```
nps 50000 j 1000
sdef
fmesh4:n geom=XYZ
        origin=-100.0,-100.0,-100.0
        imesh=100.0 iints=100
        jmesh=100.0 jints=100
        kmesh=100.0 kints=100
        out=none tally=rma_batch
```

To also test a file system's support for parallel IO, replace `out=none` with `out=xdmf` and add a `pio on` card to the end of the file.

If this feature is enabled, it will appear in `mcnp6.mpi -v` as "`mpi_thread_multiple = ON`".