SUNDIALS Installation Guide v7.3.0

SUNDIALS v7.3.0

Eddy Banks[1], Alan C. Hindmarsh[1], Radu Serban[1], Cody J. Balos[1],
David J. Gardner[1], Daniel R. Reynolds[2], and Carol S. Woodward[1]

[1]*Center for Applied Scientific Computing, Lawrence Livermore National Laboratory*
[2]*Department of Mathematics, Southern Methodist University*

April 07, 2025

UCRL-SM-208108

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

CONTRIBUTORS

The SUNDIALS library has been developed over many years by a number of contributors. The current SUNDIALS team consists of Cody J. Balos, David J. Gardner, Alan C. Hindmarsh, Daniel R. Reynolds, and Carol S. Woodward. We thank Radu Serban for significant and critical past contributions.

Other contributors to SUNDIALS include: Mustafa Aggul, James Almgren-Bell, Lawrence E. Banks, Peter N. Brown, George Byrne, Rujeko Chinomona, Scott D. Cohen, Aaron Collier, Keith E. Grant, Steven L. Lee, Shelby L. Lockhart, John Loffeld, Daniel McGreer, Yu Pan, Slaven Peles, Cosmin Petra, Steven B. Roberts, H. Hunter Schwartz, Jean M. Sexton, Dan Shumaker, Steve G. Smith, Shahbaj Sohal, Allan G. Taylor, Hilari C. Tiedeman, Chris White, Ting Yan, and Ulrike M. Yang.

Contents

Chapter 1

Installing SUNDIALS

In this chapter we discuss two ways for building and installing SUNDIALS from source. The first is with the Spack HPC package manager and the second is with CMake.

1.1 Installing with Spack

Spack is a package management tool that provides a simple spec syntax to configure and install software on a wide variety of platforms and environments. See the Getting Started section in the Spack documentation for more information on installing Spack.

Once Spack is setup on your system, the default SUNDIALS configuration can be install with the command

```
spack install sundials
```

Additional options can be enabled through various Spack package variants. For information on the available variants visit the SUNDIALS Spack package web page or use the command

```
spack info sundials
```

1.2 Installing with CMake

CMake provides a platform-independent build system capable of generating Unix and Linux Makefiles, as well as KDevelop, Visual Studio, and (Apple) XCode project files from the same configuration file. A GUI front end is also available allowing for an interactive build and installation process.

At a minimum, building SUNDIALS requires CMake version 3.18.0 or higher and a working C compiler. If a compatible version of CMake is not already installed on you system, source files or pre-built binary files can be obtained from the CMake Download website.

When building with CMake, you will need to obtain the SUNDIALS source code. You can get the source files by either cloning the SUNDIALS GitHub repository with the command

```
git clone https://github.com/LLNL/sundials
```

or by downloading release compressed archives (`.tar.gz` files) from the SUNDIALS download website. The compressed archives allow for downloading the entire SUNDIALS suite or individual packages. The name of the distribution archive is of the form `SOLVER-7.3.0.tar.gz`, where `SOLVER` is one of: `sundials`, `cvode`, `cvodes`, `arkode`, ida, idas, or kinsol, and 7.3.0 represents the version number of the SUNDIALS suite or of the individual package. After downloading the relevant archives, uncompress and expand the sources. For example, by running

```
tar -zxf SOLVER-7.3.0.tar.gz
```

the extracted source files will be under the SOLVER-7.3.0 directory.

In the installation steps below we will refer to the following directories:

- SOLVER_DIR is the sundials directory created when cloning from GitHub or the SOLVER-7.3.0 directory created after uncompressing the release archive.
- BUILD_DIR is the (temporary) directory under which SUNDIALS is built. In-source builds are prohibited; the build directory BUILD_DIR can **not** be the same as SOLVER_DIR and such an attempt will lead to an error. This prevents "polluting" the source tree, simplifies building with different configurations and/or options, and makes it easy to clean-up all traces of the build by simply removing the build directory.
- INSTALL_DIR is the directory under which the SUNDIALS exported header files and libraries will be installed. The installation directory INSTALL_DIR can not be the same as the SOLVER_DIR directory. Typically, header files are exported under a directory INSTALL_DIR/include while libraries are typically installed under INSTALL_-DIR/lib or INSTALL_LIB/lib64, with INSTALL_DIR specified at configuration time.

1.2.1 Linux/Unix systems

CMake can be used from the command line with the cmake command, or from graphical interfaces with the ccmake or cmake-gui commands. Below we present the installation steps using the command line interface.

Using CMake from the command line is simply a matter of generating the build files for the desired configuration, building, and installing. For example, the following commands will build and install the default configuration:

```
cmake \
  -S SOLVER_DIR \
  -B BUILD_DIR \
  -D CMAKE_INSTALL_PREFIX=INSTALL_DIR
cd BUILD_DIR
make
make install
```

The default configuration will install static and shared libraries for all SUNDIALS packages and install the associated example codes. Additional features can be enabled by specifying more options in the configuration step. For example, to enable MPI add -D ENABLE_MPI=ON to the cmake command above:

```
cmake \
  -S SOLVER_DIR \
  -B BUILD_DIR \
  -D CMAKE_INSTALL_PREFIX=INSTALL_DIR \
  -D ENABLE_MPI=ON
```

See section §1.3 below for a complete list of SUNDIALS configuration options and additional configuration examples.

1.2.2 Windows Systems

CMake can also be used to build SUNDIALS on Windows. To build SUNDIALS for use with Visual Studio the following steps should be performed:

1. Create a separate `BUILD_DIR`
2. Open a Visual Studio Command Prompt and cd to `BUILD_DIR`
3. Run `cmake-gui ../SOLVER_DIR`
   a. Hit Configure
   b. Check/Uncheck solvers to be built
   c. Change `CMAKE_INSTALL_PREFIX` to `INSTALL_DIR`
   d. Set other options as desired (see section §1.3)
   e. Hit Generate
4. Back in the VS Command Window:
   a. Run `msbuild ALL_BUILD.vcxproj`
   b. Run `msbuild INSTALL.vcxproj`

The resulting libraries will be in the `INSTALL_DIR`.

The SUNDIALS project can also now be opened in Visual Studio. Double click on the `ALL_BUILD.vcxproj` file to open the project. Build the whole *solution* to create the SUNDIALS libraries. To use the SUNDIALS libraries in your own projects, you must set the include directories for your project, add the SUNDIALS libraries to your project solution, and set the SUNDIALS libraries as dependencies for your project.

1.2.3 HPC Clusters

This section is a guide for installing SUNDIALS on specific HPC clusters. In general, the procedure is the same as described previously in §1.2.1 for Unix/Linux machines. The main differences are in the modules and environment variables that are specific to different HPC clusters. We aim to keep this section as up to date as possible, but it may lag the latest software updates to each cluster.

1.2.3.1 Frontier

Frontier is an Exascale supercomputer at the Oak Ridge Leadership Computing Facility. If you are new to this system, then we recommend that you review the Frontier user guide.

A Standard Installation

Load the modules and set the environment variables needed to build SUNDIALS. This configuration enables both MPI and HIP support for distributed and GPU parallelism. It uses the HIP compiler for C and C++ and the Cray Fortran compiler. Other configurations are possible.

```
# required dependencies
module load PrgEnv-cray-amd/8.5.0
module load craype-accel-amd-gfx90a
module load rocm/5.3.0
module load cmake/3.23.2

# GPU-aware MPI
```

(continues on next page)

(continued from previous page)

```
export MPICH_GPU_SUPPORT_ENABLED=1

# compiler environment hints
export CC=$(which hipcc)
export CXX=$(which hipcc)
export FC=$(which ftn)
export CFLAGS="-I${ROCM_PATH}/include"
export CXXFLAGS="-I${ROCM_PATH}/include -Wno-pass-failed"
export LDFLAGS="-L${ROCM_PATH}/lib -lamdhip64 ${PE_MPICH_GTL_DIR_amd_gfx90a} -lmpi_gtl_hsa"
```

Now we can build SUNDIALS. In general, this is the same procedure described in the previous sections. The following command builds and installs SUNDIALS with MPI, HIP, and the Fortran interface enabled, where `<account>` is your allocation account on Frontier:

```
cmake \
  -S SOLVER_DIR \
  -B BUILD_DIR \
  -D CMAKE_INSTALL_PREFIX=INSTALL_DIR \
  -D AMDGPU_TARGETS=gfx90a \
  -D ENABLE_HIP=ON \
  -D ENABLE_MPI=ON \
  -D BUILD_FORTRAN_MODULE_INTERFACE=ON
cd BUILD_DIR
make -j8 install
# Need an allocation to run the tests:
salloc -A <account> -t 10 -N 1 -p batch
make test
make test_install_all
```

1.3 Configuration options

All available SUNDIALS CMake options are described in the sections below. The default values for some options (e.g., compiler flags and installation paths) are for a Linux system and are provided as illustration only.

Note

When using a CMake graphical interface (`ccmake` or `cmake-gui`), multiple configuration passes are performed before generating the build files. For options where the default value depends on the value of another option, the initial value is set on the first configuration pass and is not updated automatically if the related option value is changed in subsequent passes. For example, the default value of *EXAMPLES_INSTALL_PATH* is `CMAKE_INSTALL_PREFIX/examples`; if the value of *CMAKE_INSTALL_PREFIX* is updated, then *EXAMPLES_INSTALL_PATH* will also need to be updated as its value was set using the *CMAKE_INSTALL_PREFIX* default.

1.3.1 Build Type

The build type determines the level of compiler optimization, if debug information is included, and if additional error checking code is generated. The provided build types are:

- `Debug` – no optimization flags, debugging information included, additional error checking enabled
- `Release` – high optimization flags, no debugging information, no additional error checks
- `RelWithDebInfo` – high optimization flags, debugging information included, no additional error checks
- `MinSizeRel` – minimize size flags, no debugging information, no additional error checks

Each build type has a corresponding option for the set of compiler flags that are appended to the user-specified compiler flags. See section §1.3.2 for more information.

`CMAKE_BUILD_TYPE`

Choose the type of build for single-configuration generators (e.g., Makefiles or Ninja).

Default: `RelWithDebInfo`

`CMAKE_CONFIGURATION_TYPES`

Specifies the build types for multi-config generators (e.g. Visual Studio, Xcode, or Ninja Multi-Config) as a semicolon-separated list.

Default: `Debug`, `Release`, `RelWithDebInfo`, and `MinSizeRel`

1.3.2 Compilers and Compiler Flags

Building SUNDIALS requires a C compiler that supports at least a subset of the C99 standard (specifically those features implemented by Visual Studio 2015).

Additional SUNDIALS features that interface with external C++ libraries or GPU programming models require a C++ compiler (e.g., CUDA, HIP, SYCL, Ginkgo, Trilinos, etc.). The C++ standard required depends on the particular library or programming model used and is noted with the relevant options below. The C++ convenience classes provided by SUNDIALS require C++14 or newer. C++ applications that require an earlier C++ standard should use the SUNDIALS C API.

When enabling the SUNDIALS Fortran interfaces, a Fortran compiler that supports the Fortran 2003 or newer standard is required in order to utilize the `ISO_C_BINDING` module.

1.3.2.1 C Compiler

`CMAKE_C_COMPILER`

The full path to the C compiler

Default: CMake will attempt to automatically locate a C compiler on the system (e.g., from the `CC` environment variable or common installation paths).

`CMAKE_C_FLAGS`

User-specified flags for the C compiler. The value of this option should be a string with flags separated by spaces.

Default: Initialized by the `CFLAGS` environment variable.

`CMAKE_C_FLAGS_DEBUG`

C compiler flags appended when the *`CMAKE_BUILD_TYPE`* is `Debug`

Default: `-g`

`CMAKE_C_FLAGS_RELEASE`

C compiler flags appended when the *`CMAKE_BUILD_TYPE`* is `Release`

Default: `-O3 -DNDEBUG`

`CMAKE_C_FLAGS_RELWITHDEBINFO`

C compiler flags appended when the *`CMAKE_BUILD_TYPE`* is `RelWithDebInfo`

Default: `-O2 -g -DNDEBUG`

`CMAKE_C_FLAGS_MINSIZEREL`

C compiler flags appended when the *`CMAKE_BUILD_TYPE`* is `MinSizeRel`

Default: `-Os -DNDEBUG`

`CMAKE_C_STANDARD`

The C standard used when building SUNDIALS C source files.

Default: `99`

Options: `99`, `11`, or `17`

`CMAKE_C_EXTENSIONS`

Enable compiler specific C extensions.

Default: `ON`

1.3.2.2 C++ Compiler

`CMAKE_CXX_COMPILER`

The full path to the C++ compiler

Default: CMake will attempt to automatically locate a C++ compiler on the system (e.g., from the `CXX` environment variable or common installation paths).

`CMAKE_CXX_FLAGS`

User-specified flags for the C++ compiler. The value of this option should be a string with flags separated by spaces.

Default: Initialized by the `CXXFLAGS` environment variable.

`CMAKE_CXX_FLAGS_DEBUG`

C++ compiler flags appended when the *`CMAKE_BUILD_TYPE`* is `Debug`

Default: `-g`

`CMAKE_CXX_FLAGS_RELEASE`

C++ compiler flags appended when the *`CMAKE_BUILD_TYPE`* is `Release`

Default: `-O3 -DNDEBUG`

`CMAKE_CXX_FLAGS_RELWITHDEBINFO`

C++ compiler flags appended when the *`CMAKE_BUILD_TYPE`* is `RelWithDebInfo`

Default: `-O2 -g -DNDEBUG`

`CMAKE_CXX_FLAGS_MINSIZEREL`

C++ compiler flags appended when the *`CMAKE_BUILD_TYPE`* is `MinSizeRel`

Default: `-Os -DNDEBUG`

`CMAKE_CXX_STANDARD`

The C++ standard used when building SUNDIALS C++ source files.

Default: `14`

Options: `14`, `17`, or `20`

`CMAKE_CXX_EXTENSIONS`

Enable compiler specific C++ extensions.

Default: `ON`

1.3.2.3 Fortran Compiler

`CMAKE_Fortran_COMPILER`

The full path to the Fortran compiler

Default: CMake will attempt to automatically locate a Fortran compiler on the system (e.g., from the `FC` environment variable or common installation paths).

`CMAKE_Fortran_FLAGS`

User-specified flags for the Fortran compiler. The value of this option should be a string with flags separated by spaces.

Default: Initialized by the `FFLAGS` environment variable.

`CMAKE_Fortran_FLAGS_DEBUG`

Fortran compiler flags appended when the *`CMAKE_BUILD_TYPE`* is `Debug`

Default: `-g`

`CMAKE_Fortran_FLAGS_RELEASE`

Fortran compiler flags appended when the *`CMAKE_BUILD_TYPE`* is `Release`

Default: `-O3`

`CMAKE_Fortran_FLAGS_RELWITHDEBINFO`

Fortran compiler flags appended when the *`CMAKE_BUILD_TYPE`* is `RelWithDebInfo`

Default: `-O2 -g`

`CMAKE_Fortran_FLAGS_MINSIZEREL`

Fortran compiler flags appended when the *`CMAKE_BUILD_TYPE`* is `MinSizeRel`

Default: `-Os`

1.3.3 Install Location

Use the following options to set where the SUNDIALS headers, library, and CMake configuration files will be installed.

`CMAKE_INSTALL_PREFIX`

Install path prefix (`INSTALL_DIR`), prepended onto install directories

Default: `/usr/local`

Note

The user must have write access to the location specified through this option. Exported SUNDIALS header files and libraries will be installed under subdirectories `include` and *`CMAKE_INSTALL_LIBDIR`* of *`CMAKE_INSTALL_PREFIX`*, respectively.

`CMAKE_INSTALL_LIBDIR`

The directory under *`CMAKE_INSTALL_PREFIX`* where libraries will be installed

Default: Set based on the system as `lib`, `lib64`, or `lib/<multiarch-tuple>`

`SUNDIALS_INSTALL_CMAKEDIR`

The directory under *`CMAKE_INSTALL_PREFIX`* where the SUNDIALS CMake package configuration files will be installed (see section §1.6.1 for more information)

Default: `CMAKE_INSTALL_LIBDIR/cmake/sundials`

1.3.4 Shared and Static Libraries

Use the following options to set which types of libraries will be installed. By default both static and shared libraries are installed.

`BUILD_SHARED_LIBS`

Build shared libraries

Default: `ON`

`BUILD_STATIC_LIBS`

Build static libraries

Default: `ON`

1.3.5 Index Size

`SUNDIALS_INDEX_SIZE`

The integer size (in bits) used for indices in SUNDIALS (e.g., for vector and matrix entries), options are: `32` or `64`

Default: `64`

Note

The build system tries to find an integer type of the appropriate size. Candidate 64-bit integer types are (in order of preference): `int64_t`, `__int64`, `long long`, and `long`. Candidate 32-bit integers are (in order of preference): `int32_t`, `int`, and `long`. The advanced option, *`SUNDIALS_INDEX_TYPE`* can be used to provide a type not listed here.

`SUNDIALS_INDEX_TYPE`

The integer type used for SUNDIALS indices. The type size must match the size provided in the *`SUNDIALS_INDEX_SIZE`* option.

Default: Automatically determined based on *`SUNDIALS_INDEX_SIZE`*

Changed in version 3.2.0: In prior versions, this option could be set to `INT64_T` to use 64-bit integers or `INT32_T` to use 32-bit integers. These special values are deprecated and a user will only need to use the *`SUNDIALS_INDEX_SIZE`* option in most cases.

1.3.6 Precision

`SUNDIALS_PRECISION`

The floating-point precision used in SUNDIALS packages and class implementations, options are: `single`, `double`, or `extended`

Default: `double`

1.3.7 Math Library

`SUNDIALS_MATH_LIBRARY`

The standard C math library (e.g., `libm`) to link with.

Default: `-lm` on Unix systems, none otherwise

1.3.8 SUNDIALS Packages

The following options can be used to enable/disable particular SUNDIALS packages.

`BUILD_ARKODE`

Build the ARKODE library

Default: `ON`

`BUILD_CVODE`

Build the CVODE library

Default: `ON`

`BUILD_CVODES`

Build the CVODES library

Default: `ON`

`BUILD_IDA`

Build the IDA library

Default: `ON`

`BUILD_IDAS`

Build the IDAS library

Default: `ON`

`BUILD_KINSOL`

Build the KINSOL library

Default: `ON`

1.3.9 Example Programs

`EXAMPLES_ENABLE_C`

Build the SUNDIALS C examples

Default: `ON`

`EXAMPLES_ENABLE_CXX`

Build the SUNDIALS C++ examples

Default: `OFF`

`EXAMPLES_ENABLE_CUDA`

Build the SUNDIALS CUDA examples

Default: `ON` when *`ENABLE_CUDA`* is `ON`, otherwise `OFF`

`EXAMPLES_ENABLE_F2003`

Build the SUNDIALS Fortran 2003 examples

Default: `ON` when *`BUILD_FORTRAN_MODULE_INTERFACE`* is `ON`, otherwise `OFF`

`EXAMPLES_INSTALL`

Install example program source files and sample output files. See *`EXAMPLES_INSTALL_PATH`* for the install location.

A `CMakeLists.txt` file to build the examples will be automatically generated and installed with the source files. If building on a Unix-like system, a `Makefile` for compiling the installed example programs will be also generated and installed.

Default: `ON`

`EXAMPLES_INSTALL_PATH`

Full path to where example source and output files will be installed

Default: `CMAKE_INSTALL_PREFIX/examples`

1.3.10 Fortran Interfaces

`BUILD_FORTRAN_MODULE_INTERFACE`

Build the SUNDIALS Fortran 2003 interface

Default: `OFF`

Note

The Fortran interface are only compatible with double precision (i.e., *`SUNDIALS_PRECISION`* must be `double`).

Warning

There is a known issue with MSYS/gfortran and SUNDIALS shared libraries that causes linking the Fortran interfaces to fail when building SUNDIALS. For now the work around is to only build with static libraries when using MSYS with gfortran on Windows.

1.3.11 Error Checking

For more information on error handling in SUNDIALS, see Error Checking.

`SUNDIALS_ENABLE_ERROR_CHECKS`

Build SUNDIALS with more extensive checks for unrecoverable errors.

Default: `ON` when *`CMAKE_BUILD_TYPE`* is `Debug`, otherwise `OFF`

Warning

Error checks will impact performance, but can be helpful for debugging.

1.3.12 Logging

For more information on logging in SUNDIALS, see Status and Error Logging.

`SUNDIALS_LOGGING_LEVEL`

The maximum logging level. The options are:

- `0` – no logging
- `1` – log errors
- `2` – log errors + warnings
- `3` – log errors + warnings + informational output
- `4` – log errors + warnings + informational output + debug output
- `5` – log all of the above and even more (e.g. vector valued variables may be logged)

Default: `2`

Warning

Logging will impact performance, but can be helpful for debugging or understanding algorithm performance. The higher the logging level, the more output that may be logged, and the more performance may degrade.

1.3.13 Monitoring

`SUNDIALS_BUILD_WITH_MONITORING`

Build SUNDIALS with capabilities for fine-grained monitoring of solver progress and statistics. This is primarily useful for debugging.

Default: `OFF`

Warning

Building with monitoring may result in minor performance degradation even if monitoring is not utilized.

1.3.14 Profiling

For more information on profiling in SUNDIALS, see Performance Profiling.

SUNDIALS_BUILD_WITH_PROFILING

Build SUNDIALS with capabilities for fine-grained profiling. This requires POSIX timers, the Windows `profileapi.h` timers, or enabling Caliper with *ENABLE_CALIPER*.

Default: `OFF`

Warning

Profiling will impact performance, and should be enabled judiciously.

1.3.15 Building with Adiak

Adiak is a library for recording meta-data about HPC simulations. Adiak is developed by Lawrence Livermore National Laboratory and can be obtained from the Adiak GitHub repository.

ENABLE_ADIAK

Enable Adiak support

Default: `OFF`

adiak_DIR

Path to the root of an Adiak installation

Default: None

1.3.16 Building with Caliper

Caliper is a performance analysis library providing a code instrumentation and performance measurement framework for HPC applications. Caliper is developed by Lawrence Livermore National Laboratory and can be obtained from the Caliper GitHub repository.

When profiling and Caliper are both enabled, SUNDIALS will utilize Caliper for performance profiling.

To enable Caliper support, set the *ENABLE_CALIPER* to `ON` and set *CALIPER_DIR* to the root path of the Caliper installation. For example, the following command will configure SUNDIALS with profiling and Caliper support:

```
cmake \
  -S SOLVER_DIR \
  -B BUILD_DIR \
  -D CMAKE_INSTALL_PREFIX=INSTALL_DIR \
  -D SUNDIALS_BUILD_WITH_PROFILING=ON \
  -D ENABLE_CALIPER=ON \
  -D CALIPER_DIR=/path/to/caliper/installation
```

ENABLE_CALIPER

Enable CALIPER support

Default: `OFF`

Note

Using Caliper requires setting *SUNDIALS_BUILD_WITH_PROFILING* to ON.

CALIPER_DIR

Path to the root of a Caliper installation

Default: None

1.3.17 Building with CUDA

The NVIDIA CUDA Toolkit provides a development environment for GPU-accelerated computing with NVIDIA GPUs. The CUDA Toolkit and compatible NVIDIA drivers are available from the NVIDIA developer website. SUNDIALS has been tested with the CUDA toolkit versions 10, 11, and 12.

When CUDA support is enabled, the CUDA NVector, the cuSPARSE SUNMatrix, and the cuSPARSE batched QR SUNLinearSolver will be built (see sections §1.7.3.11, §1.7.4.2, and §1.7.5.2, respectively, for the corresponding header files and libraries). For more information on using SUNDIALS with GPUs, see Features for GPU Accelerated Computing.

To enable CUDA support, set *ENABLE_CUDA* to ON. If CUDA is installed in a nonstandard location, you may need to set *CUDA_TOOLKIT_ROOT_DIR* to your CUDA Toolkit installation path. You will also need to set *CMAKE_CUDA_ARCHITECTURES* to the CUDA architecture for your system. For example, the following command will configure SUNDIALS with CUDA support for a system with an Ampere GPU:

```
cmake \
  -S SOLVER_DIR \
  -B BUILD_DIR \
  -D CMAKE_INSTALL_PREFIX=INSTALL_DIR \
  -D ENABLE_CUDA=ON \
  -D CMAKE_CUDA_ARCHITECTURES="80"
```

ENABLE_CUDA

Enable CUDA support

Default: OFF

CUDA_TOOLKIT_ROOT_DIR

Path to the CUDA Toolkit installation

Default: CMake will attempt to automatically locate an installed CUDA Toolkit

CMAKE_CUDA_ARCHITECTURES

Specifies the CUDA architecture to compile for i.e., 60 for Pascal, 70 for Volta, 80 for Ampere, 90 for Hopper, etc. See the GPU compute capability tables on the NVIDIA webpage and the GPU Compilation section of the CUDA documentation for more information.

Default: Determined automatically by CMake. Users are encouraged to override this value with the architecture for their system as the default varies across compilers and compiler versions.

Changed in version 7.2.0: In prior versions *CMAKE_CUDA_ARCHITECTURES* defaulted to 70.

1.3.18 Building with Ginkgo

Ginkgo is a high-performance linear algebra library with a focus on solving sparse linear systems. It is implemented using modern C++ (you will need at least a C++14 compliant compiler to build it), with GPU kernels implemented in CUDA (for NVIDIA devices), HIP (for AMD devices), and SYCL/DPC++ (for Intel devices and other supported hardware). Ginkgo can be obtained from the Ginkgo GitHub repository. SUNDIALS is regularly tested with the latest versions of Ginkgo, specifically up to version 1.8.0.

When Ginkgo support is enabled, the Ginkgo SUNMatrix and the Ginkgo SUNLinearSolver header files will be installed (see sections §1.7.4.4 and §1.7.5.4, respectively, for the corresponding header files). For more information on using SUNDIALS with GPUs, see Features for GPU Accelerated Computing.

To enable Ginkgo support, set *ENABLE_GINKGO* to `ON` and set *Ginkgo_DIR* to the root path of the Ginkgo installation. Additionally, set *SUNDIALS_GINKGO_BACKENDS* to a semicolon-separated list of Ginkgo target architectures/executors. For example, the following command will configure SUNDIALS with Ginkgo support using the reference, OpenMP, and CUDA (targeting Ampere GPUs) backends:

```
cmake \
  -S SOLVER_DIR \
  -B BUILD_DIR \
  -D CMAKE_INSTALL_PREFIX=INSTALL_DIR \
  -D ENABLE_GINKGO=ON \
  -D Ginkgo_DIR=/path/to/ginkgo/installation \
  -D SUNDIALS_GINKGO_BACKENDS="REF;OMP;CUDA" \
  -D ENABLE_CUDA=ON \
  -D CMAKE_CUDA_ARCHITECTURES="80" \
  -D ENABLE_OPENMP=ON
```

Note

The SUNDIALS interfaces to Ginkgo are not compatible with extended precision (i.e., when *SUNDIALS_PRECISION* is set to `extended`).

`ENABLE_GINKGO`

Enable Ginkgo support

Default: `OFF`

`Ginkgo_DIR`

Path to the Ginkgo installation

Default: None

`SUNDIALS_GINKGO_BACKENDS`

Semi-colon separated list of Ginkgo target architectures/executors to build for. Options currently supported are `REF` (the Ginkgo reference executor), `OMP` (OpenMP), `CUDA`, `HIP`, and `SYCL`.

Default: `"REF;OMP"`

Changed in version 7.1.0: The `DPCPP` option was changed to `SYCL` to align with Ginkgo's naming convention.

1.3.19 Building with HIP

The Heterogeneous-compute Interface for Portability (HIP) allows developers to create portable applications for AMD and NVIDIA GPUs. HIP can be obtained from the HIP GitHub repository. SUNDIALS has been tested with HIP versions between 5.0.0 to 5.4.3.

When HIP support is enabled, the HIP NVector will be built (see section §1.7.3.12 for the corresponding header file and library). For more information on using SUNDIALS with GPUs, see Features for GPU Accelerated Computing.

To enable HIP support, set *ENABLE_HIP* to ON and set *AMDGPU_TARGETS* to the desired target (e.g., gfx705). In addition, set *CMAKE_C_COMPILER* and *CMAKE_CXX_COMPILER* to a HIP compatible compiler e.g., hipcc. For example, the following command will configure SUNDIALS with HIP support for a system with an MI250X GPU:

```
cmake \
  -S SOLVER_DIR \
  -B BUILD_DIR \
  -D CMAKE_INSTALL_PREFIX=INSTALL_DIR \
  -D CMAKE_C_COMPILER=hipcc \
  -D CMAKE_CXX_COMPILER=hipcc \
  -D ENABLE_HIP=ON \
  -D AMDGPU_TARGETS="gfx90a"
```

ENABLE_HIP

Enable HIP Support

Default: OFF

AMDGPU_TARGETS

Specify which AMD GPUs to target

Default: None

1.3.20 Building with *hypre*

hypre is a library of high performance preconditioners and solvers featuring multigrid methods for the solution of large, sparse linear systems of equations on massively parallel computers. The library is developed by Lawrence Livermore National Laboratory and is available from the hypre GitHub repository. SUNDIALS is regularly tested with the latest versions of *hypre*, specifically up to version 2.26.0.

When *hypre* support is enabled, the ParHyp NVector will be built (see section §1.7.3.9 for the corresponding header file and library).

To enable *hypre* support, set *ENABLE_MPI* to ON, set *ENABLE_HYPRE* to ON, and set *HYPRE_DIR* to the root path of the *hypre* installation. For example, the following command will configure SUNDIALS with *hypre* support:

```
cmake \
  -S SOLVER_DIR \
  -B BUILD_DIR \
  -D CMAKE_INSTALL_PREFIX=INSTALL_DIR \
  -D ENABLE_MPI=ON \
  -D ENABLE_HYPRE=ON \
  -D HYPRE_DIR=/path/to/hypre/installation
```

Note

SUNDIALS must be configured so that *SUNDIALS_INDEX_SIZE* is compatible with HYPRE_BigInt in the *hypre* installation.

ENABLE_HYPRE

Enable *hypre* support

Default: OFF

HYPRE_DIR

Path to the *hypre* installation

Default: none

1.3.21 Building with KLU

KLU is a software package for the direct solution of sparse nonsymmetric linear systems of equations that arise in circuit simulation and is part of SuiteSparse, a suite of sparse matrix software. The library is developed by Texas A&M University and is available from the SuiteSparse GitHub repository. SUNDIALS is regularly tested with the latest versions of KLU, specifically up to SuiteSparse version 7.7.0.

When KLU support is enabled, the KLU SUNLinearSolver will be built (see section §1.7.5.5 for the corresponding header file and library).

To enable KLU support, set *ENABLE_KLU* to ON. For SuiteSparse 7.4.0 and newer, set *KLU_ROOT* to the root of the SuiteSparse installation. Alternatively, set *KLU_INCLUDE_DIR* and *KLU_LIBRARY_DIR* to the path to the header and library files, respectively, of the SuiteSparse installation. For example, the following command will configure SUNDIALS with KLU support:

```
cmake \
  -S SOLVER_DIR \
  -B BUILD_DIR \
  -D CMAKE_INSTALL_PREFIX=INSTALL_DIR \
  -D ENABLE_KLU=ON \
  -D KLU_ROOT=/path/to/suitesparse/installation
```

ENABLE_KLU

Enable KLU support

Default: OFF

KLU_ROOT

Path to the SuiteSparse installation

Default: OFF

KLU_INCLUDE_DIR

Path to SuiteSparse header files

Default: none

KLU_LIBRARY_DIR

Path to SuiteSparse installed library files

Default: none

1.3.22 Building with Kokkos

Kokkos is a modern C++ (requires at least C++14) programming model for witting performance portable code for multicore CPU and GPU-based systems including NVIDIA, AMD, and Intel GPUs. Kokkos is developed by Sandia National Laboratory and can be obtained from the Kokkos GitHub repository. The minimum supported version of Kokkos 3.7.00. SUNDIALS is regularly tested with the latest versions of Kokkos, specifically up to version 4.3.01.

When Kokkos support is enabled, the Kokkos NVector header file will be installed (see section §1.7.3.16 for the corresponding header file). For more information on using SUNDIALS with GPUs, see Features for GPU Accelerated Computing.

To enable Kokkos support, set the *ENABLE_KOKKOS* to `ON` and set *Kokkos_DIR* to root path of the Kokkos installation. For example, the following command will configure SUNDIALS with Kokkos support:

```
cmake \
  -S SOLVER_DIR \
  -B BUILD_DIR \
  -D CMAKE_INSTALL_PREFIX=INSTALL_DIR \
  -D ENABLE_KOKKOS=ON \
  -D Kokkos_DIR=/path/to/kokkos/installation
```

ENABLE_KOKKOS

Enable Kokkos support

Default: `OFF`

Kokkos_DIR

Path to the Kokkos installation.

Default: None

1.3.23 Building with KokkosKernels

The KokkosKernels library is built on Kokkos and provides common linear algebra computational kernels. KokkosKernels is developed by Sandia National Laboratory and can be obtained from the KokkosKernels GitHub repository. The minimum supported version of KokkosKernels 3.7.00. SUNDIALS is regularly tested with the latest versions of KokkosKernels, specifically up to version 4.3.01.

When KokkosKernels support is enabled, the KokkosKernels SUNMatrix and KokkosKernels SUNLinearSolver header files will be installed (see sections §1.7.4.5 and §1.7.5.6, respectively, for the corresponding header files). For more information on using SUNDIALS with GPUs, see Features for GPU Accelerated Computing.

To enable KokkosKernels support, set *ENABLE_KOKKOS* and *ENABLE_KOKKOS_KERNELS* to `ON` and set *Kokkos_DIR* and *KokkosKernels_DIR* to the root paths for the Kokkos and KokkosKernels installations, respectively. For example, the following command will configure SUNDIALS with Kokkos and KokkosKernels support:

```
cmake \
  -S SOLVER_DIR \
  -B BUILD_DIR \
  -D CMAKE_INSTALL_PREFIX=INSTALL_DIR \
  -D ENABLE_KOKKOS=ON \
  -D Kokkos_DIR=/path/to/kokkos/installation \
  -D ENABLE_KOKKOS_KERNELS=ON \
  -D KokkosKernels_DIR=/path/to/kokkoskernels/installation
```

ENABLE_KOKKOS_KERNELS

Enable KokkosKernels support

Default: `OFF`

KokkosKernels_DIR

Path to the KokkosKernels installation.

Default: None

1.3.24 Building with LAPACK

The Linear Algebra PACKage (LAPACK) library interface defines functions for solving systems of linear equations. Several LAPACK implementations are available e.g., the Netlib reference implementation, the Intel oneAPI Math Kernel Library, or OpenBLAS (among others). SUNDIALS is regularly tested with the latest versions of OpenBLAS, specifically up to version 0.3.27.

When LAPACK support is enabled, the LAPACK banded SUNLinearSolver and LAPACK dense SUNLinearSolver will be built (see sections §1.7.5.7 and §1.7.5.8, respectively, for the corresponding header files and libraries).

To enable LAPACK support, set *ENABLE_LAPACK* to `ON`. CMake will attempt to find BLAS and LAPACK installations on the system and set the variables *BLAS_LIBRARIES*, *BLAS_LINKER_FLAGS*, *LAPACK_LIBRARIES*, and *LAPACK_-LINKER_FLAGS*. To explicitly specify the LAPACK library to build with, manually set the aforementioned variables to the desired values when configuring the build. For example, the following command will configure SUNDIALS with LAPACK support:

```
cmake \
  -S SOLVER_DIR \
  -B BUILD_DIR \
  -D CMAKE_INSTALL_PREFIX=INSTALL_DIR \
  -D ENABLE_LAPACK=ON \
  -D BLAS_LIBRARIES=/path/to/lapack/installation/lib/libblas.so \
  -D LAPACK_LIBRARIES=/path/to/lapack/installation/lib/liblapack.so
```

Note

If a working Fortran compiler is not available to infer the name-mangling scheme for LAPACK functions, the options *SUNDIALS_LAPACK_CASE* and *SUNDIALS_LAPACK_UNDERSCORES* *must* be set to bypass the check for a Fortran compiler and define the name-mangling scheme. The defaults for these options in earlier versions of SUNDIALS were `lower` and `one`, respectively.

ENABLE_LAPACK

Enable LAPACK support

Default: `OFF`

BLAS_LIBRARIES

BLAS libraries

Default: none (CMake will try to find a BLAS installation)

BLAS_LINKER_FLAGS

BLAS required linker flags

Default: none (CMake will try to determine the necessary flags)

LAPACK_LIBRARIES

LAPACK libraries

Default: none (CMake will try to find a LAPACK installation)

LAPACK_LINKER_FLAGS

LAPACK required linker flags

Default: none (CMake will try to determine the necessary flags)

SUNDIALS_LAPACK_CASE

Specify the case to use in the Fortran name-mangling scheme, options are: `lower` or `upper`

Default:

> **Note**
>
> The build system will attempt to infer the Fortran name-mangling scheme using the Fortran compiler. This option should only be used if a Fortran compiler is not available or to override the inferred or default (`lower`) scheme if one can not be determined. If used, *SUNDIALS_LAPACK_UNDERSCORES* must also be set.

SUNDIALS_LAPACK_UNDERSCORES

Specify the number of underscores to append in the Fortran name-mangling scheme, options are: `none`, `one`, or `two`

Default:

> **Note**
>
> The build system will attempt to infer the Fortran name-mangling scheme using the Fortran compiler. This option should only be used if a Fortran compiler is not available or to override the inferred or default (`one`) scheme if one can not be determined. If used, *SUNDIALS_LAPACK_CASE* must also be set.

1.3.25 Building with MAGMA

The Matrix Algebra on GPU and Multicore Architectures (MAGMA) project provides a dense linear algebra library similar to LAPACK but targeting heterogeneous architectures. The library is developed by the University of Tennessee and is available from the MAGMA GitHub repository. SUNDIALS is regularly tested with the latest versions of MAGMA, specifically up to version 2.8.0.

When MAGMA support is enabled, the MAGMA dense SUNMatrix and MAGMA dense SUNLinearSolver will be built (see sections §1.7.4.6 and §1.7.5.9, respectively, for the corresponding header files and libraries). For more information on using SUNDIALS with GPUs, see Features for GPU Accelerated Computing.

To enable MAGMA support, set *ENABLE_MAGMA* to `ON`, *MAGMA_DIR* to the root path of MAGMA installation, and *SUNDIALS_MAGMA_BACKENDS* to the desired MAGMA backend to use. For example, the following command will configure SUNDIALS with MAGMA support with the CUDA backend (targeting Ampere GPUs):

```
cmake \
  -S SOLVER_DIR \
  -B BUILD_DIR \
  -D CMAKE_INSTALL_PREFIX=INSTALL_DIR \
  -D ENABLE_MAGMA=ON \
  -D MAGMA_DIR=/path/to/magma/installation \
```

(continues on next page)

(continued from previous page)

```
  -D SUNDIALS_MAGMA_BACKEND="CUDA" \
  -D ENABLE_CUDA=ON \
  -D CMAKE_CUDA_ARCHITECTURES="80"
```

ENABLE_MAGMA

Enable MAGMA support

Default: OFF

MAGMA_DIR

Path to the MAGMA installation

Default: none

SUNDIALS_MAGMA_BACKENDS

Which MAGMA backend to use under the SUNDIALS MAGMA interface: CUDA or HIP

Default: CUDA

1.3.26 Building with MPI

The Message Passing Interface (MPI) is a standard for communication on parallel computing systems. Several MPI implementations are available e.g., OpenMPI, MPICH, MVAPICH, Cray MPICH, Intel MPI, or IBM Spectrum MPI (among others). SUNDIALS is regularly tested with the latest versions of OpenMPI, specifically up to version 5.0.5.

When MPI support is enabled, the parallel NVector, MPI ManyVector NVector, and MPI+X NVector will be built (see sections §1.7.3.3, §1.7.3.4, and §1.7.3.5, respectively, for the corresponding header files and libraries).

Attention

Changed in version 7.0.0: When MPI is enabled, all SUNDIALS libraries will include MPI symbols and applications will need to include the path for MPI headers and link against the corresponding MPI library.

To enable MPI support, set *ENABLE_MPI* to ON. If CMake is unable to locate an MPI installation, set the relevant MPI_<language>_COMPILER options to the desired MPI compilers. For example, the following command will configure SUNDIALS with MPI support:

```
cmake \
  -S SOLVER_DIR \
  -B BUILD_DIR \
  -D CMAKE_INSTALL_PREFIX=INSTALL_DIR \
  -D ENABLE_MPI=ON
```

ENABLE_MPI

Enable MPI support

Default: OFF

MPI_C_COMPILER

The MPI C compiler e.g., mpicc

Default: CMake will attempt to locate an MPI C compiler

MPI_CXX_COMPILER

The MPI C++ compiler e.g., `mpicxx`

Default: CMake will attempt to locate an MPI C++ compiler

Note

This option is only needed if MPI is enabled (*ENABLE_MPI* is `ON`) and C++ examples are enabled (*EXAMPLES_ENABLE_CXX* is `ON`). All SUNDIALS solvers can be used from C++ MPI applications by without setting any additional configuration options other than *ENABLE_MPI*.

MPI_Fortran_COMPILER

The MPI Fortran compiler e.g., `mpif90`

Default: CMake will attempt to locate an MPI Fortran compiler

Note

This option is triggered only needed if MPI is enabled (*ENABLE_MPI* is `ON`) and the Fortran interfaces are enabled (*BUILD_FORTRAN_MODULE_INTERFACE* is `ON`).

MPIEXEC_EXECUTABLE

Specify the executable for running MPI programs e.g., `mpiexec`

Default: CMake will attempt to locate the MPI executable

MPIEXEC_PREFLAGS

Specifies flags that come directly after `MPIEXEC_EXECUTABLE` and before `MPIEXEC_NUMPROC_FLAG` and `MPIEXEC_MAX_NUMPROCS`.

Default: none

MPIEXEC_POSTFLAGS

Specifies flags that come after the executable to run but before any other program arguments.

Default: none

1.3.27 Building with oneMKL

The Intel oneAPI Math Kernel Library (oneMKL) includes CPU and SYCL/DPC++ interfaces for LAPACK dense linear algebra routines. The SUNDIALS oneMKL interface targets the SYCL/DPC++ routines, to utilize the CPU routine see section §1.3.24. SUNDIALS has been tested with oneMKL version 2021.4.

When oneMKL support is enabled, the oneMLK dense SUNMatrix and the oneMKL dense SUNLinearSolver will be built (see sections §1.7.4.7 and §1.7.5.10, respectively, for the corresponding header files and libraries). For more information on using SUNDIALS with GPUs, see Features for GPU Accelerated Computing.

To enable the SUNDIALS oneMKL interface set *ENABLE_ONEMKL* to `ON` and *ONEMKL_DIR* to the root path of oneMKL installation. For example, the following command will configure SUNDIALS with oneMKL support:

```
cmake \
  -S SOLVER_DIR \
  -B BUILD_DIR \
  -D CMAKE_INSTALL_PREFIX=INSTALL_DIR \
```

(continues on next page)

(continued from previous page)

```
  -D ENABLE_ONEMKL=ON \
  -D ONEMKL_DIR=/path/to/onemkl/installation \
```

ENABLE_ONEMKL

Enable oneMKL support

Default: OFF

ONEMKL_DIR

Path to oneMKL installation.

Default: none

SUNDIALS_ONEMKL_USE_GETRF_LOOP

This advanced debugging option replaces the batched LU factorization with a loop over each system in the batch and a non-batched LU factorization.

Default: OFF

SUNDIALS_ONEMKL_USE_GETRS_LOOP

This advanced debugging option replaces the batched LU solve with a loop over each system in the batch and a non-batched solve.

Default: OFF

1.3.28 Building with OpenMP

The OpenMP API defines a directive-based approach for portable parallel programming across architectures.

When OpenMP support is enabled, the OpenMP NVector will be built (see section §1.7.3.6 for the corresponding header file and library).

To enable OpenMP support, set the *ENABLE_OPENMP* to ON. For example, the following command will configure SUNDIALS with OpenMP support:

```
cmake \
  -S SOLVER_DIR \
  -B BUILD_DIR \
  -D CMAKE_INSTALL_PREFIX=INSTALL_DIR \
  -D ENABLE_OPENMP=ON
```

ENABLE_OPENMP

Enable OpenMP support

Default: OFF

1.3.29 Building with OpenMP Device Offloading

The OpenMP 4.0 specification added support for offloading computations to devices (i.e., GPUs). SUNDIALS requires OpenMP 4.5 for GPU offloading support.

When OpenMP offloading support is enabled, the OpenMPDEV NVector will be built (see section §1.7.3.7 for the corresponding header file and library).

To enable OpenMP device offloading support, set the *ENABLE_OPENMP_DEVICE* to `ON`. For example, the following command will configure SUNDIALS with OpenMP device offloading support:

```
cmake \
  -S SOLVER_DIR \
  -B BUILD_DIR \
  -D CMAKE_INSTALL_PREFIX=INSTALL_DIR \
  -D ENABLE_OPENMP_DEVICE=ON
```

ENABLE_OPENMP_DEVICE

Enable OpenMP device offloading support

Default: `OFF`

1.3.30 Building with PETSc

The Portable, Extensible Toolkit for Scientific Computation (PETSc) is a suite of data structures and routines for simulating applications modeled by partial differential equations. The library is developed by Argonne National Laboratory and is available from the PETSc GitLab repository. SUNDIALS requires PETSc 3.5.0 or newer and is regularly tested with the latest versions of PETSc, specifically up to version 3.21.4.

When PETSc support is enabled, the PETSc NVector and PETSc SNES SUNNonlinearSolver will be built (see sections §1.7.3.10 and §1.7.6.3, respectively, for the corresponding header files and libraries).

To enable PETSc support, set *ENABLE_MPI* to `ON`, set *ENABLE_PETSC* to `ON`, and set *PETSC_DIR* to the path of the PETSc installation. Alternatively, a user can provide a list of include paths in *PETSC_INCLUDES* and a list of complete paths to the PETSc libraries in *PETSC_LIBRARIES*. For example, the following command will configure SUNDIALS with PETSc support:

```
cmake \
  -S SOLVER_DIR \
  -B BUILD_DIR \
  -D CMAKE_INSTALL_PREFIX=INSTALL_DIR \
  -D ENABLE_MPI=ON \
  -D ENABLE_PETSC=ON \
  -D PETSC_DIR=/path/to/petsc/installation
```

ENABLE_PETSC

Enable PETSc support

Default: `OFF`

PETSC_DIR

Path to PETSc installation

Default: none

PETSC_LIBRARIES

Semi-colon separated list of PETSc link libraries. Unless provided by the user, this is autopopulated based on the PETSc installation found in *PETSC_DIR*.

Default: none

PETSC_INCLUDES

Semi-colon separated list of PETSc include directories. Unless provided by the user, this is autopopulated based on the PETSc installation found in *PETSC_DIR*.

Default: none

1.3.31 Building with PThreads

POSIX Threads (PThreads) is an API for shared memory programming defined by the Institute of Electrical and Electronics Engineers (IEEE) standard POSIX.1c.

When PThreads support is enabled, the PThreads NVector will be built (see section §1.7.3.8 for the corresponding header file and library).

To enable PThreads support, set *ENABLE_PTHREAD* to ON. For example, the following command will configure SUNDIALS with PThreads support:

```
cmake \
  -S SOLVER_DIR \
  -B BUILD_DIR \
  -D CMAKE_INSTALL_PREFIX=INSTALL_DIR \
  -D ENABLE_PTHREAD=ON
```

ENABLE_PTHREAD

Enable PThreads support

Default: OFF

1.3.32 Building with RAJA

RAJA is a performance portability layer developed by Lawrence Livermore National Laboratory and can be obtained from the RAJA GitHub repository. SUNDIALS is regularly tested with the latest versions of RAJA, specifically up to version 2024.02.2.

When RAJA support is enabled, the RAJA NVector will be built (see section §1.7.3.13 for the corresponding header files and libraries).

To enable RAJA support, set *ENABLE_RAJA* to ON, set *RAJA_DIR* to the path of the RAJA installation, set *SUNDIALS_RAJA_BACKENDS* to the desired backend (CUDA, HIP, or SYCL), and set *ENABLE_CUDA*, *ENABLE_HIP*, or *ENABLE_SYCL* to ON depending on the selected backend. For example, the following command will configure SUNDIALS with RAJA support using the CUDA backend (targeting Ampere GPUs):

```
cmake \
  -S SOLVER_DIR \
  -B BUILD_DIR \
  -D CMAKE_INSTALL_PREFIX=INSTALL_DIR \
  -D ENABLE_RAJA=ON \
  -D RAJA_DIR=/path/to/raja/installation \
  -D SUNDIALS_RAJA_BACKENDS="CUDA" \
```

(continues on next page)

(continued from previous page)

```
  -D ENABLE_CUDA=ON \
  -D CMAKE_CUDA_ARCHITECTURES="80"
```

ENABLE_RAJA

Enable RAJA support

Default: OFF

RAJA_DIR

Path to the RAJA installation

Default: none

SUNDIALS_RAJA_BACKENDS

If building SUNDIALS with RAJA support, this sets the RAJA backend to target. Values supported are CUDA, HIP, or SYCL.

Default: CUDA

1.3.33 Building with SuperLU_DIST

SuperLU_DIST is a general purpose library for the direct solution of large, sparse, nonsymmetric systems of linear equations in a distributed memory setting. The library is developed by Lawrence Berkeley National Laboratory and is available from the SuperLU_DIST GitHub repository. SuperLU_DIST version 7.0.0 or newer is required. SUNDIALS is regularly tested with the latest versions of SuperLU_DIST, specifically up to version 8.2.1.

When SuperLU_DIST support is enabled, the SuperLU_DIST (SLUNRloc) SUNMatrix and SuperLU_DIST SUNLinearSolver will be built (see sections §1.7.4.9 and §1.7.5.16 for the corresponding header files and libraries).

To enable SuperLU_DIST support, set *ENABLE_MPI* to ON, set *ENABLE_SUPERLUDIST* to ON, and set *SUPERLUDIST_DIR* to the path where SuperLU_DIST is installed. If SuperLU_DIST was built with OpenMP enabled, set *SUPERLUDIST_OpenMP* and *ENABLE_OPENMP* to ON. For example, the following command will configure SUNDIALS with SuperLU_DIST support:

```
cmake \
  -S SOLVER_DIR \
  -B BUILD_DIR \
  -D CMAKE_INSTALL_PREFIX=INSTALL_DIR \
  -D ENABLE_SUPERLUDIST=ON \
  -D SUPERLUDIST_DIR=/path/to/superludist/installation
```

ENABLE_SUPERLUDIST

Enable SuperLU_DIST support

Default: OFF

SUPERLUDIST_DIR

Path to SuperLU_DIST installation.

Default: none

SUPERLUDIST_OpenMP

Enable SUNDIALS support for SuperLU_DIST built with OpenMP

Default: none

Note

SuperLU_DIST must be built with OpenMP support for this option to function. Additionally the environment variable `OMP_NUM_THREADS` must be set to the desired number of threads.

`SUPERLUDIST_INCLUDE_DIRS`

List of include paths for SuperLU_DIST (under a typical SuperLU_DIST install, this is typically the SuperLU_-DIST `SRC` directory)

Default: none

Note

This is an advanced option. Prefer to use *`SUPERLUDIST_DIR`*.

`SUPERLUDIST_LIBRARIES`

Semi-colon separated list of libraries needed for SuperLU_DIST

Default: none

Note

This is an advanced option. Prefer to use *`SUPERLUDIST_DIR`*.

`SUPERLUDIST_INCLUDE_DIR`

Path to SuperLU_DIST header files (under a typical SuperLU_DIST install, this is typically the SuperLU_DIST `SRC` directory)

Default: none

Note

This is an advanced option. This option is deprecated. Use *`SUPERLUDIST_INCLUDE_DIRS`*.

`SUPERLUDIST_LIBRARY_DIR`

Path to SuperLU_DIST installed library files

Default: none

Note

This option is deprecated. Use *`SUPERLUDIST_DIR`*.

1.3.34 Building with SuperLU_MT

SuperLU_MT is a general purpose library for the direct solution of large, sparse, nonsymmetric systems of linear equations on shared memory parallel machines. The library is developed by Lawrence Berkeley National Laboratory and is available from the SuperLU_MT GitHub repository. SUNDIALS is regularly tested with the latest versions of SuperLU_MT, specifically up to version 4.0.1.

When SuperLU_MT support is enabled, the SuperLU_MT SUNLinearSolver will be built (see section §1.7.5.17 for the corresponding header file and library).

To enable SuperLU_MT support, set *ENABLE_SUPERLUMT* to `ON`, set *SUPERLUMT_INCLUDE_DIR* and *SUPERLUMT_-LIBRARY_DIR* to the location of the header and library files, respectively, of the SuperLU_MT installation. Depending on the SuperLU_MT installation, it may also be necessary to set *SUPERLUMT_LIBRARIES* to a semi-colon separated list of other libraries SuperLU_MT depends on. For example, if SuperLU_MT was build with an external blas library, then include the full path to the blas library in this list. Additionally, the variable *SUPERLUMT_THREAD_TYPE* must be set to either `Pthread` or `OpenMP`. For example, the following command will configure SUNDIALS with SuperLU_MT support using PThreads:

```
cmake \
  -S SOLVER_DIR \
  -B BUILD_DIR \
  -D CMAKE_INSTALL_PREFIX=INSTALL_DIR \
  -D ENABLE_SUPERLUMT=ON \
  -D SUPERLUMT_INCLUDE_DIR=/path/to/superlumt/installation/include/dir \
  -D SUPERLUMT_LIBRARY_DIR=/path/to/superlumt/installation/library/dir \
  -D SUPERLUMT_THREAD_TYPE="Pthread"
```

Warning

Do not mix thread types when using SUNDIALS packages. For example, if using the OpenMP or PThreads NVector then the SuperLU_MT installation should use the same threading type.

`ENABLE_SUPERLUMT`

Enable SuperLU_MT support

Default: `OFF`

`SUPERLUMT_INCLUDE_DIR`

Path to SuperLU_MT header files (under a typical SuperLU_MT install, this is typically the SuperLU_MT `SRC` directory)

Default: none

`SUPERLUMT_LIBRARY_DIR`

Path to SuperLU_MT installed library files

Default: none

`SUPERLUMT_LIBRARIES`

Semi-colon separated list of libraries needed for SuperLU_MT

Default: none

`SUPERLUMT_THREAD_TYPE`

Must be set to Pthread or OpenMP, depending on how SuperLU_MT was compiled.

Default: Pthread

1.3.35 Building with SYCL

SYCL is an abstraction layer for programming heterogeneous parallel computing based on C++17.

When SYCL support is enabled, the SYCL NVector will be built (see section §1.7.3.14 for the corresponding header file and library).

To enable SYCL support, set the *ENABLE_SYCL* to ON. For example, the following command will configure SUNDIALS with SYCL support using Intel compilers:

```
cmake \
  -S SOLVER_DIR \
  -B BUILD_DIR \
  -D CMAKE_INSTALL_PREFIX=INSTALL_DIR \
  -D CMAKE_C_COMPILER=icx \
  -D CMAKE_CXX_COMPILER=icpx \
  -D CMAKE_CXX_FLAGS="-fsycl" \
  -D ENABLE_SYCL=ON
```

ENABLE_SYCL

Enable SYCL support

Default: OFF

Note

Building with SYCL enabled requires a compiler that supports a subset of the of SYCL 2020 specification (specifically `sycl/sycl.hpp` must be available).

CMake does not currently support autodetection of SYCL compilers and *CMAKE_CXX_COMPILER* must be set to a valid SYCL compiler. At present the only supported SYCL compilers are the Intel oneAPI compilers i.e., `dpcpp` and `icpx`. When using `icpx` the `-fsycl` flag and any ahead of time compilation flags must be added to *CMAKE_CXX_FLAGS*.

SUNDIALS_SYCL_2020_UNSUPPORTED

This advanced option disables the use of *some* features from the SYCL 2020 standard in SUNDIALS libraries and examples. This can be used to work around some cases of incomplete compiler support for SYCL 2020.

Default: OFF

1.3.36 Building with Trilinos

Trilinos is a collection of C++ libraries of linear solvers, non-linear solvers, optimization solvers, etc. developed by Sandia National Laboratory and available from the Trilinos GitHub repository. SUNDIALS is regularly tested with the latest versions of Trilinos, specifically up to version 16.0.0.

When Trilinos support is enabled, the Trilinos Tpetra NVector will be built (see section §1.7.3.15 for the corresponding header file and library).

To enable Trilinos support, set the *ENABLE_TRILINOS* to ON and set *Trilinos_DIR* to root path of the Trilinos installation. For example, the following command will configure SUNDIALS with Trilinos support:

```
cmake \
  -S SOLVER_DIR \
  -B BUILD_DIR \
```

(continues on next page)

(continued from previous page)

```
  -D CMAKE_INSTALL_PREFIX=INSTALL_DIR \
  -D ENABLE_TRILONOS=ON \
  -D TRILINOS_DIR=/path/to/trilinos/installation
```

ENABLE_TRILINOS

Enable Trilinos support

Default: OFF

Trilinos_DIR

Path to the Trilinos installation

Default: None

1.3.37 Building with XBraid

XBraid is parallel-in-time library implementing an optimal-scaling multigrid reduction in time (MGRIT) solver. The library is developed by Lawrence Livermore National Laboratory and is available from the XBraid GitHub repository. SUNDIALS is regularly tested with the latest versions of XBraid, specifically up to version 3.0.0.

To enable XBraid support, set *ENABLE_MPI* to ON, set *ENABLE_XBRAID* to ON, set *XBRAID_DIR* to the root path of the XBraid installation. For example, the following command will configure SUNDIALS with XBraid support:

```
cmake \
  -S SOLVER_DIR \
  -B BUILD_DIR \
  -D CMAKE_INSTALL_PREFIX=INSTALL_DIR \
  -D SUNDIALS_INDEX_SIZE="32" \
  -D ENABLE_MPI=ON \
  -D ENABLE_XBRAID=ON \
  -D XBRAID_DIR=/path/to/xbraid/installation
```

Note

At this time the XBraid types `braid_Int` and `braid_Real` are hard-coded to `int` and `double` respectively. As such SUNDIALS must be configured with *SUNDIALS_INDEX_SIZE* set to 32 and *SUNDIALS_PRECISION* set to `double`. Additionally, SUNDIALS must be configured with *ENABLE_MPI* set to ON.

ENABLE_XBRAID

Enable or disable the ARKStep + XBraid interface.

Default: OFF

XBRAID_DIR

The root directory of the XBraid installation.

Default: OFF

XBRAID_INCLUDES

Semi-colon separated list of XBraid include directories. Unless provided by the user, this is autopopulated based on the XBraid installation found in *XBRAID_DIR*.

Default: none

XBRAID_LIBRARIES

Semi-colon separated list of XBraid link libraries. Unless provided by the user, this is autopopulated based on the XBraid installation found in *XBRAID_DIR*.

Default: none

1.3.38 Building with xSDK Defaults

The Extreme-scale Scientific Software Development Kit (xSDK) is a community of HPC libraries and applications developing best practices and standards for scientific software.

USE_XSDK_DEFAULTS

Enable xSDK default configuration settings. This sets the default value for *CMAKE_BUILD_TYPE* to `Debug`, *SUNDIALS_INDEX_SIZE* to `32`, and *SUNDIALS_PRECISION* to `double`.

Default: `OFF`

1.3.39 Building with External Addons

SUNDIALS "addons" are community developed code additions for SUNDIALS that can be subsumed by the SUNDIALS build system so that they have full access to all internal SUNDIALS symbols. The intent is for SUNDIALS addons to function as if they are part of the SUNDIALS library, while allowing them to potentially have different licenses (although we encourage BSD-3-Clause still), code style (although we encourage them to follow the SUNDIALS style outlined here).

Warning

SUNDIALS addons are not maintained by the SUNDIALS team and may come with different licenses. Use them at your own risk.

To build with SUNDIALS addons,

1. Clone/copy the addon(s) into `SOLVER_DIR/external/`
2. Copy the `sundials-addon-example` block in the `SOLVER_DIR/external/CMakeLists.txt`, paste it below the example block, and modify the path listed for your own external addon(s).
3. When building SUNDIALS, set the CMake option *SUNDIALS_ENABLE_EXTERNAL_ADDONS* to `ON`
4. Build SUNDIALS as usual.

SUNDIALS_ENABLE_EXTERNAL_ADDONS

Build SUNDIALS with any external addons that you have put in `SOLVER_DIR/external`.

Default: `OFF`

1.4 Testing the Build and Installation

If SUNDIALS was configured with any `EXAMPLES_ENABLE_<language>` options set to `ON`, then a set of regression tests can be run after building with the command:

```
make test
```

Additionally, if *`EXAMPLES_INSTALL`* is set to `ON`, then a set of smoke tests can be run after installing with the command:

```
make test_install
```

1.5 Building and Running Examples

Each of the SUNDIALS solvers is distributed with a set of examples demonstrating basic usage. To build and install the examples, set at least one of the `EXAMPLES_ENABLE_<language>` options to `ON`, and set *`EXAMPLES_INSTALL`* to `ON`. Along side the example sources and outputs, automatically generated `CMakeLists.txt` configuration files (and `Makefile` files if on Linux/Unix systems) are installed referencing the *installed* SUNDIALS headers and libraries.

Either the `CMakeLists.txt` file or the traditional `Makefile` may be used to build the examples and serve as a template for building user developed problems. To use the supplied `Makefile` simply run `make` to compile and generate the executables. To use CMake from within the installed example directory, run `cmake` (or `ccmake` or `cmake-gui` to use the GUI) followed by `make` to compile the example code. Note that if CMake is used, it will overwrite the traditional `Makefile` with a new CMake-generated `Makefile`.

The resulting output from running the examples can be compared with example output bundled in the SUNDIALS distribution.

Note

There will potentially be differences in the output due to machine architecture, compiler versions, use of third party libraries, etc.

1.6 Using SUNDIALS In Your Project

After installing SUNDIALS, building your application with SUNDIALS involves two steps: including the right header files and linking to the right libraries. Depending on what features of SUNDIALS that your application uses, the header files and libraries needed will vary. For example, if you want to use CVODE for serial computations you need the following includes:

```
#include <cvode/cvode.h>
#include <nvector/nvector_serial.h>
```

and must link to `libsundials_cvode` and `libsundials_nvecserial`. If you wanted to use CVODE with the GMRES linear solver and the CUDA NVector, you need the following includes:

```
#include <cvode/cvode.h>
#include <nvector/nvector_cuda.h>
#include <sunlinsol/sunlinsol_spgmr.h>
```

and must link to `libsundials_cvode`, `libsundials_nveccuda`, and `libsundials_sunlinsolspgmr`.

Attention

Added in version 7.0.0: All applications must also link to `libsundials_core`. For projects using SUNDIALS CMake targets (see section §1.6.1), this dependency is automatically included.

Refer to section §1.7 below or the documentations sections for the individual SUNDIALS packages and modules of interest for the proper includes and libraries to link against.

1.6.1 CMake Projects

For projects that use CMake, the SUNDIALS CMake package configuration file provides CMake targets for the consuming project. Use the CMake `find_package` command to search for the configuration file, `SUNDIALSConfig.cmake`, which is installed alongside a package version file, `SUNDIALSConfigVersion.cmake`, under the `INSTALL_DIR/SUNDIALS_INSTALL_CMAKEDIR` directory. The SUNDIALS CMake targets follow the same naming convention as the generated library binaries with the `libsundials_` prefix replaced by `SUNDIALS::`. For example, the exported target for `libsundials_cvode` is `SUNDIALS::cvode`. See section §1.7 for a complete list of CMake targets. The CMake code snippit below shows how a consuming project might leverage the SUNDIALS package configuration file to build against SUNDIALS in their own CMake project.

```
project(MyProject)

# Set the variable SUNDIALS_DIR to the SUNDIALS instdir.
# When using the cmake CLI command, this can be done like so:
#   cmake -D SUNDIALS_DIR=/path/to/sundials/installation

# Find any SUNDIALS version...
find_package(SUNDIALS REQUIRED)

# ... or find any version newer than some minimum...
find_package(SUNDIALS 7.1.0 REQUIRED)

# ... or find a version in a range
find_package(SUNDIALS 7.0.0...7.1.0 REQUIRED)

# To check if specific components are available in the SUNDIALS installation,
# use the COMPONENTS option followed by the desired target names
find_package(SUNDIALS REQUIRED COMPONENTS cvode nvecpetsc)

add_executable(myexec main.c)

# Link to SUNDIALS libraries through the exported targets.
# This is just an example, users should link to the targets appropriate
# for their use case.
target_link_libraries(myexec PUBLIC SUNDIALS::cvode SUNDIALS::nvecpetsc)
```

Note

Changed in version 7.1.0: A single version provided to `find_package` denotes the minimum version of SUNDIALS to look for, and any version equal or newer than what is specified will match. In prior versions `SUNDIALSConfig.cmake` required the version found to have the same major version number as the single version provided to `find_package`.

To accommodate installing both static and shared libraries simultaneously, targets are created with `_static` and `_shared` suffixes, respectively, and the un-suffixed target is an alias to the `_shared` version. For example, `SUNDIALS::cvode` is an alias to `SUNDIALS::cvode_shared` in this case. Projects that wish to use static libraries should use the `_static` version of the target when both library types are installed. When only static or shared libraries (not both) are installed the un-suffixed alias corresponds to the library type chosen at configuration time (see section §1.3.4).

1.7 Libraries and Header Files

As noted above, the SUNDIALS the header files and libraries are installed under the *`CMAKE_INSTALL_PREFIX`* path in the `include` and *`CMAKE_INSTALL_LIBDIR`* subdirectories, respectively. The public header files are further organized into subdirectories under the `include` directory. The installed public header files and libraries are listed for reference in the sections below. Additionally, the exported CMake targets are also listed for projects using CMake (see section §1.6.1). The file extension `.LIB` used below is typically `.so`, `.dll`, or `.dylib` for shared libraries and `.a` or `.lib` for static libraries.

Warning

SUNDIALS installs some header files to `CMAKE_INSTALL_PREFIX/include/sundials/priv`. All of the header files in this directory are private and **should not be included in user code**. The private headers are subject to change without any notice and relying on them may break your code.

1.7.1 SUNDIALS Core

The core library contains the shared infrastructure utilized by SUNDIALS packages. All applications using SUNDIALS must link against the core library. For codes using the SUNDIALS CMake targets, the core target is automatically included as needed by other targets.

Table 1.1: The SUNDIALS core library, header, and CMake target

Libraries	`libsundials_core.LIB`
Headers	`sundials/sundials_core.h`
CMake target	`SUNDIALS::core`

The core header file is a convenient way to include all the header files that make up the SUNDIALS core infrastructure.

Table 1.2: Header files included by `sundials_core.h`

Headers	`sundials/sundials_adaptcontroller.h`
	`sundials/sundials_adjointstepper.h`
	`sundials/sundials_adjointcheckpointscheme.h`
	`sundials/sundials_config.h`
	`sundials/sundials_context.h`
	`sundials/sundials_errors.h`
	`sundials/sundials_iterative.h`
	`sundials/sundials_linearsolver.h`
	`sundials/sundials_logger.h`
	`sundials/sundials_math.h`
	`sundials/sundials_matrix.h`
	`sundials/sundials_memory.h`
	`sundials/sundials_nonlinearsolver.h`
	`sundials/sundials_nvector.h`
	`sundials/sundials_profiler.h`
	`sundials/sundials_types.h`
	`sundials/sundials_version.h`

For C++ applications, several convenience classes are provided for interacting with SUNDIALS objects. These can be accessed by including the C++ core header file.

Table 1.3: The SUNDIALS C++ core header file

Headers	`sundials/sundials_core.hpp`

Like the C core header file, the C++ core header file is a convenient way to include all the header files for the core C++ classes.

Warning

Features in the `sundials::experimental` namespace are not yet part of the public API and are subject to change or removal without notice.

Table 1.4: Header files included by `sundials_core.hpp`

Headers	`sundials/sundials_context.hpp`
	`sundials/sundials_core.h`
	`sundials/sundials_linearsolver.hpp`
	`sundials/sundials_matrix.hpp`
	`sundials/sundials_memory.hpp`
	`sundials/sundials_nonlinearsolver.hpp`
	`sundials/sundials_nvector.hpp`
	`sundials/sundials_profiler.hpp`

When MPI support is enabled (*`ENABLE_MPI`* is `ON`), the following header file provides aliases between MPI data types and SUNDIALS types. The alias `MPI_SUNREALTYPE` is one of `MPI_FLOAT`, `MPI_DOUBLE`, or `MPI_LONG_DOUBLE` depending on the value of *`SUNDIALS_PRECISION`*. The alias `MPI_SUNINDEXTYPE` is either `MPI_INT32_T` or `MPI_-INT64_T` depending on the value of *`SUNDIALS_INDEX_SIZE`*.

Table 1.5: Header file defining aliases between SUNDIALS and MPI data types

Headers	`sundials/sundials_mpi_types.h`

When XBraid support is enabled (*ENABLE_XBRAID* is `ON`), the following header file defines types and functions for interfacing SUNDIALS with XBraid.

Table 1.6: SUNDIALS header for interfacing with XBraid

Headers	`sundials/sundials_xbraid.h`

1.7.2 SUNDIALS Packages

1.7.2.1 CVODE

To use the CVODE package, include the header file and link to the library given below.

Table 1.7: CVODE library, header file, and CMake target

Libraries	`libsundials_cvode.LIB`
Headers	`cvode/cvode.h`
CMake target	`SUNDIALS::cvode`

The CVODE header file includes the files below which define functions, types, and constants for the CVODE linear solver interface and using projection methods with CVODE.

Table 1.8: Additional header files included by `cvode.h`

Headers	`cvode/cvode_ls.h`
	`cvode/cvode_proj.h`

CVODE provides a specialized linear solver module for diagonal linear systems. Include the header file below to access the related functions.

Table 1.9: CVODE diagonal linear solver

Headers	`cvode/cvode_diag.h`

For problems in which the user cannot define a more effective, problem-specific preconditioner for Krylov iterative linear solvers, CVODE provides banded (`bandpre`) and band-block-diagonal (`bbdpre`) preconditioner modules. Include the header files below to access the related functions.

Table 1.10: CVODE preconditioner modules

Headers	`cvode/cvode_bandpre.h`
	`cvode/cvode_bbdpre.h`

1.7.2.2 CVODES

To use the CVODES package, include the header file and link to the library given below.

Warning

CVODES is a superset of CVODE and defines the same functions as provided by CVODE. As such, applications should not link to both CVODES and CVODE.

Table 1.11: CVODES library, header file, and CMake target

Libraries	`libsundials_cvodes.LIB`
Headers	`cvodes/cvodes.h`
CMake target	`SUNDIALS::cvodes`

The CVODES header file includes the files below which define functions, types, and constants for the CVODES linear solver interface and using projection methods with CVODES.

Table 1.12: Additional header files included by `cvodes.h`

Headers	`cvodes/cvodes_ls.h`
	`cvodes/cvodes_proj.h`

CVODES provides a specialized linear solver module for diagonal linear systems. Include the header file below to access the related functions.

Table 1.13: CVODES diagonal linear solver

Headers	`cvodes/cvodes_diag.h`

For problems in which the user cannot define a more effective, problem-specific preconditioner for Krylov iterative linear solvers, CVODES provides banded (`bandpre`) and band-block-diagonal (`bbdpre`) preconditioner modules. Include the header files below to access the related functions.

Table 1.14: CVODES preconditioner modules

Headers	`cvodes/cvodes_bandpre.h`
	`cvodes/cvodes_bbdpre.h`

1.7.2.3 ARKODE

To use the ARKODE package, link to the library below and include the header file for the desired module.

Table 1.15: ARKODE library, header files, and CMake target

Libraries	libsundials_arkode.LIB
Headers	arkode/arkode_arkstep.h
	arkode/arkode_erkstep.h
	arkode/arkode_forcingstep.h
	arkode/arkode_lsrkstep.h
	arkode/arkode_mristep.h
	arkode/arkode_splittingstep.h
	arkode/arkode_sprkstep.h
CMake target	SUNDIALS::arkode

The ARKODE module header files include the header file for the shared ARKODE interface functions, constants, and types (`arkode.h`). As appropriate, the module header files also include the ARKODE linear solver interface as well as the header files defining method coefficients.

Table 1.16: Additional header files included by `arkode_*step.h` header files

Headers	arkode/arkode.h
	arkode/arkode_butcher.h
	arkode/arkode_butcher_dirk.h
	arkode/arkode_butcher_erk.h
	arkode/arkode_ls.h
	arkode/arkode_sprk.h

For problems in which the user cannot define a more effective, problem-specific preconditioner for Krylov iterative linear solvers, ARKODE provides banded (`bandpre`) and band-block-diagonal (`bbdpre`) preconditioner modules. Include the header files below to access the related functions.

Table 1.17: ARKODE preconditioner modules

Headers	arkode/arkode_bandpre.h
	arkode/arkode_bbdpre.h

When XBraid support is enabled (*`ENABLE_XBRAID`* is `ON`), include the ARKODE-XBraid interface header file and link to the interface library given below to use ARKODE and XBraid together.

Table 1.18: ARKODE library, header, and CMake target for interfacing with XBraid

Libraries	libsundials_arkode_xbraid.LIB
Headers	arkode/arkode_xbraid.h
CMake target	SUNDIALS::arkode_xbraid

1.7.2.4 IDA

To use the IDA package, include the header file and link to the library given below.

Table 1.19: IDA library, header file, and CMake target

Libraries	`libsundials_ida.LIB`
Headers	`ida/ida.h`
CMake target	`SUNDIALS::ida`

The IDA header file includes the header file below which defines functions, types, and constants for the IDA linear solver interface.

Table 1.20: Additional header files included by `ida.h`

Headers	`ida/ida_ls.h`

For problems in which the user cannot define a more effective, problem-specific preconditioner for Krylov iterative linear solvers, IDA provides a band-block-diagonal (`bbdpre`) preconditioner module. Include the header file below to access the related functions.

Table 1.21: IDA preconditioner modules

Headers	`ida/ida_bbdpre.h`

1.7.2.5 IDAS

To use the IDAS package, include the header file and link to the library given below.

Warning

IDAS is a superset of IDA and defines the same functions as provided by IDA. As such, applications should not link to both IDAS and IDA.

Table 1.22: IDAS library, header file, and CMake target

Libraries	`libsundials_idas.LIB`
Headers	`idas/idas.h`
CMake target	`SUNDIALS::idas`

The IDAS header file includes the header file below which defines functions, types, and constants for the IDAS linear solver interface.

Table 1.23: Additional header files included by `idas.h`

Headers	`idas/idas_ls.h`

For problems in which the user cannot define a more effective, problem-specific preconditioner for Krylov iterative linear solvers, IDAS provides a band-block-diagonal (`bbdpre`) preconditioner module. Include the header file below to access the related functions.

Table 1.24: IDAS preconditioner modules

Headers	`idas/idas_bbdpre.h`

1.7.2.6 KINSOL

To use the KINSOL package, include the header file and link to the library given below.

Table 1.25: KINSOL library, header file, and CMake target

Libraries	`libsundials_kinsol.LIB`
Headers	`kinsol/kinsol.h`
CMake target	`SUNDIALS::kinsol`

The KINSOL header file includes the header file below which defines functions, types, and constants for the KINSOL linear solver interface.

Table 1.26: Additional header files included by `kinsol.h`

Headers	`kinsol/kinsol_ls.h`

For problems in which the user cannot define a more effective, problem-specific preconditioner for Krylov iterative linear solvers, KINSOL provides a band-block-diagonal (`bbdpre`) preconditioner module. Include the header file below to access the related functions.

Table 1.27: KINSOL preconditioner modules

Headers	`kinsol/kinsol_bbdpre.h`

1.7.3 Vectors

1.7.3.1 Serial

To use the serial NVector, include the header file and link to the library given below.

When using SUNDIALS time integration packages or the KINSOL package, the serial NVector is bundled with the package library and it is not necessary to link to the library below when using those packages.

Table 1.28: The serial NVector library, header file, and CMake target

Libraries	`libsundials_nvecserial.LIB`
Headers	`nvector/nvector_serial.h`
CMake target	`SUNDIALS::nvecserial`

1.7.3.2 ManyVector

To use the ManyVector NVector, include the header file and link to the library given below.

Table 1.29: The ManyVector NVector library, header file, and CMake target

Libraries	`libsundials_nvecmanyvector.LIB`
Headers	`nvector/nvector_manyvector.h`
CMake target	`SUNDIALS::nvecmanyvector`

1.7.3.3 Parallel (MPI)

To use the parallel (MPI) NVector, include the header file and link to the library given below.

Table 1.30: The parallel (MPI) NVector library, header file, and CMake target

Libraries	`libsundials_nvecparallel.LIB`
Headers	`nvector/nvector_parallel.h`
CMake target	`SUNDIALS::nvecparallel`

1.7.3.4 MPI ManyVector

To use the MPI ManyVector NVector, include the header file and link to the library given below.

Table 1.31: The MPI ManyVector NVector library, header file, and CMake target

Libraries	`libsundials_nvecmpimanyvector.LIB`
Headers	`nvector/nvector_mpimanyvector.h`
CMake target	`SUNDIALS::nvecmpimanyvector`

1.7.3.5 MPI+X

To use the MPI+X NVector, include the header file and link to the library given below.

Table 1.32: The MPI+X NVector library, header file, and CMake target

Libraries	`libsundials_nvecmpiplusx.LIB`
Headers	`nvector/nvector_mpiplusx.h`
CMake target	`SUNDIALS::nvecmpiplusx`

1.7.3.6 OpenMP

To use the OpenMP NVector, include the header file and link to the library given below.

Table 1.33: The OpenMP NVector library, header file, and CMake target

Libraries	`libsundials_nvecopenmp.LIB`
Headers	`nvector/nvector_openmp.h`
CMake target	`SUNDIALS::nvecopenmp`

1.7.3.7 OpenMPDEV

To use the OpenMP device offload NVector, include the header file and link to the library given below.

Table 1.34: The OpenMP device offload NVector library, header file, and CMake target

Libraries	`libsundials_nvecopenmpdev.LIB`
Headers	`nvector/nvector_openmpdev.h`
CMake target	`SUNDIALS::nvecopenmpdev`

1.7.3.8 PThreads

To use the POSIX Threads NVector, include the header file and link to the library given below.

Table 1.35: The POSIX Threads NVector library, header file, and CMake target

Libraries	`libsundials_nvecpthreads.LIB`
Headers	`nvector/nvector_pthreads.h`
CMake target	`SUNDIALS::nvecpthreads`

1.7.3.9 *hypre* (ParHyp)

To use the hypre (ParHyp) NVector, include the header file and link to the library given below.

Table 1.36: The *hypre* (ParHyp) NVector library, header file, and CMake target

Libraries	`libsundials_nvecparhyp.LIB`
Headers	`nvector/nvector_parhyp.h`
CMake target	`SUNDIALS::nvecparhyp`

1.7.3.10 PETSc

To use the PETSc NVector, include the header file and link to the library given below.

Table 1.37: The PETSc NVector library, header file, and CMake target

Libraries	`libsundials_nvecpetsc.LIB`
Headers	`nvector/nvector_petsc.h`
CMake target	`SUNDIALS::nvecpetsc`

1.7.3.11 CUDA

To use the CUDA NVector, include the header file and link to the library given below.

Table 1.38: The CUDA NVector library, header file, and CMake target

Libraries	`libsundials_nveccuda.LIB`
Headers	`nvector/nvector_cuda.h`
CMake target	`SUNDIALS::nveccuda`

1.7.3.12 HIP

To use the HIP NVector, include the header file and link to the library given below.

Table 1.39: The HIP NVector library, header file, and CMake target

Libraries	`libsundials_nvechip.LIB`
Headers	`nvector/nvector_hip.h`
CMake target	`SUNDIALS::nvechip`

1.7.3.13 RAJA

To use the RAJA NVector, include the header file and link to the library given below for the desired backend.

Table 1.40: The RAJA NVector libraries, header file, and CMake targets

Libraries	`libsundials_nveccudaraja.LIB`
	`libsundials_nvechipraja.LIB`
	`libsundials_nvecsyclraja.LIB`
Headers	`nvector/nvector_raja.h`
CMake target	`SUNDIALS::nveccudaraja`
	`SUNDIALS::nvechipraja`
	`SUNDIALS::nvecsyclraja`

1.7.3.14 SYCL

To use the SYCL NVector, include the header file and link to the library given below.

Table 1.41: The SYCL NVector library, header file, and CMake target

Libraries	`libsundials_nvecsycl.LIB`
Headers	`nvector/nvector_sycl.h`
CMake target	`SUNDIALS::nvecsycl`

1.7.3.15 Trilinos (Tpetra)

To use the Trilinos (Tpetra) NVector, include the header file and link to the library given below.

Table 1.42: The Trilinos (Tpetra) NVector library, header file, and CMake target

Libraries	`libsundials_nvectrilinos.LIB`
Headers	`nvector/nvector_trilinos.h`
CMake target	`SUNDIALS::nvectrilinos`

1.7.3.16 Kokkos

To use the Kokkos NVector, include the header file and link to the library given below.

Table 1.43: The Kokkos NVector library, header file, and CMake target

Headers	`nvector/nvector_kokkos.hpp`
CMake target	`SUNDIALS::nveckokkos`

1.7.4 Matrices

1.7.4.1 Banded

To use the banded SUNMatrix, include the header file and link to the library given below.

When using SUNDIALS time integration packages or the KINSOL package, the banded SUNMatrix is bundled with the package library and it is not necessary to link to the library below when using those packages.

Table 1.44: The banded SUNMatrix library, header file, and CMake target

Libraries	`libsundials_sunmatrixband.LIB`
Headers	`sunmatrix/sunmatrix_band.h`
CMake target	`SUNDIALS::sunmatrixband`

1.7.4.2 cuSPARSE

To use the cuSPARSE SUNMatrix, include the header file and link to the library given below.

Table 1.45: The cuSPARSE SUNMatrix library, header file, and CMake target

Libraries	`libsundials_sunmatrixcusparse.LIB`
Headers	`sunmatrix/sunmatrix_cusparse.h`
CMake target	`SUNDIALS::sunmatrixcusparse`

1.7.4.3 Dense

To use the dense SUNMatrix, include the header file and link to the library given below.

When using SUNDIALS time integration packages or the KINSOL package, the dense SUNMatrix is bundled with the package library and it is not necessary to link to the library below when using those packages.

Table 1.46: The dense SUNMatrix library, header file, and CMake target

Libraries	`libsundials_sunmatrixdense.LIB`
Headers	`sunmatrix/sunmatrix_dense.h`
CMake target	`SUNDIALS::sunmatrixdense`

1.7.4.4 Ginkgo

To use the Ginkgo SUNMatrix, include the header file given below.

Table 1.47: The Ginkgo SUNMatrix library, header file, and CMake target

Headers	`sunmatrix/sunmatrix_ginkgo.hpp`
CMake target	`SUNDIALS::sunmatrixginkgo`

1.7.4.5 KokkosKernels Dense

To use the KokkosKernels dense SUNMatrix, include the header file given below.

Table 1.48: The dense KokkosKernels SUNMatrix library, header file, and CMake target

Headers	`sunmatrix/sunmatrix_kokkosdense.hpp`
CMake target	`SUNDIALS::sunmatrixkokkosdense`

1.7.4.6 MAGMA Dense

To use the MAGMA dense SUNMatrix, include the header file and link to the library given below.

Table 1.49: The dense MAGMA SUNMatrix library, header file, and CMake target

Libraries	`libsundials_sunmatrixmagmadense.LIB`
Headers	`sunmatrix/sunmatrix_magmadense.h`
CMake target	`SUNDIALS::sunmatrixmagmadense`

1.7.4.7 oneMKL Dense

To use the oneMKL dense SUNMatrix, include the header file and link to the library given below.

Table 1.50: The dense oneMKL SUNMatrix library, header file, and CMake target

Libraries	`libsundials_sunmatrixonemkldense.LIB`
Headers	`sunmatrix/sunmatrix_onemkldense.h`
CMake target	`SUNDIALS::sunmatrixonemkldense`

1.7.4.8 Sparse

To use the sparse SUNMatrix, include the header file and link to the library given below.

When using SUNDIALS time integration packages or the KINSOL package, the sparse SUNMatrix is bundled with the package library and it is not necessary to link to the library below when using those packages.

Table 1.51: The sparse SUNMatrix library, header file, and CMake target

Libraries	`libsundials_sunmatrixsparse.LIB`
Headers	`sunmatrix/sunmatrix_sparse.h`
CMake target	`SUNDIALS::sunmatrixsparse`

1.7.4.9 SuperLU_DIST (SLUNRloc)

To use the SuperLU_DIST (SLUNRloc) SUNMatrix, include the header file and link to the library given below.

Table 1.52: The SuperLU_DIST (SLUNRloc) SUNMatrix library, header file, and CMake target

Libraries	`libsundials_sunmatrixslunrloc.LIB`
Headers	`sunmatrix/sunmatrix_slunrloc.h`
CMake target	`SUNDIALS::sunmatrixslunrloc`

1.7.5 Linear Solvers

1.7.5.1 Banded

To use the banded SUNLinearSolver, include the header file and link to the library given below.

When using SUNDIALS time integration packages or the KINSOL package, the banded SUNLinearSolver is bundled with the package library and it is not necessary to link to the library below when using those packages.

Table 1.53: The banded SUNLinearSolver library, header file, and CMake target

Libraries	`libsundials_sunlinsolband.LIB`
Headers	`sunlinsol/sunlinsol_band.h`
CMake target	`SUNDIALS::sunlinsolband`

1.7.5.2 cuSPARSE Batched QR

To use the cuSPARSE batched QR SUNLinearSolver, include the header file and link to the library given below.

Table 1.54: The cuSPARSE batched QR SUNLinearSolver library, header file, and CMake target

Libraries	`libsundials_sunlinsolcusolversp.LIB`
Headers	`sunlinsol/sunlinsol_cusolversp_batchqr.h`
CMake target	`SUNDIALS::sunlinsolcusolversp`

1.7.5.3 Dense

To use the dense SUNLinearSolver, include the header file and link to the library given below.

When using SUNDIALS time integration packages or the KINSOL package, the dense SUNLinearSolver is bundled with the package library and it is not necessary to link to the library below when using those packages.

Table 1.55: The dense SUNLinearSolver library, header file, and CMake target

Libraries	`libsundials_sunlinsoldense.LIB`
Headers	`sunlinsol/sunlinsol_dense.h`
CMake target	`SUNDIALS::sunlinsoldense`

1.7.5.4 Ginkgo

To use the Ginkgo SUNLinearSolver, include the header file given below.

Table 1.56: The Ginkgo SUNLinearSolver header file and CMake target

Headers	`sunlinsol/sunlinsol_ginkgo.hpp`
CMake target	`SUNDIALS::sunlinsolginkgo`

1.7.5.5 KLU

To use the KLU SUNLinearSolver, include the header file and link to the library given below.

Table 1.57: The KLU SUNLinearSolver library, header file, and CMake target

Libraries	`libsundials_sunlinsolklu.LIB`
Headers	`sunlinsol/sunlinsol_klu.h`
CMake target	`SUNDIALS::sunlinsolklu`

1.7.5.6 KokkosKernels Dense

To use the KokkosKernels dense SUNLinearSolver, include the header file given below.

Table 1.58: The KokkosKernels dense SUNLinearSolver header file and CMake target

Headers	`sunlinsol/sunlinsol_kokkosdense.hpp`
CMake target	`SUNDIALS::sunlinsolkokkosdense`

1.7.5.7 LAPACK Banded

To use the LAPACK banded SUNLinearSolver, include the header file and link to the library given below.

Table 1.59: The LAPACK banded SUNLinearSolver library, header file, and CMake target

Libraries	`libsundials_sunlinsollapackband.LIB`
Headers	`sunlinsol/sunlinsol_lapackband.h`
CMake target	`SUNDIALS::sunlinsollapackband`

1.7.5.8 LAPACK Dense

To use the LAPACK dense SUNLinearSolver, include the header file and link to the library given below.

Table 1.60: The LAPACK dense SUNLinearSolver library, header file, and CMake target

Libraries	`libsundials_sunlinsollapackdense.LIB`
Headers	`sunlinsol/sunlinsol_lapackdense.h`
CMake target	`SUNDIALS::sunlinsollapackdense`

1.7.5.9 MAGMA Dense

To use the MAGMA dense SUNLinearSolver, include the header file and link to the library given below.

Table 1.61: The MAGMA dense SUNLinearSolver library, header file, and CMake target

Libraries	`libsundials_sunlinsolmagmadense.LIB`
Headers	`sunlinsol/sunlinsol_magmadense.h`
CMake target	`SUNDIALS::sunlinsolmagmadense`

1.7.5.10 oneMKL Dense

To use the oneMKL dense SUNLinearSolver, include the header file and link to the library given below.

Table 1.62: The oneMKL dense SUNLinearSolver library, header file, and CMake target

Libraries	`libsundials_sunlinsolonemkldense.LIB`
Headers	`sunlinsol/sunlinsol_onemkldense.h`
CMake target	`SUNDIALS::sunlinsolonemkldense`

1.7.5.11 Preconditioned Conjugate Gradient (PCG)

To use the PCG SUNLinearSolver, include the header file and link to the library given below.

When using SUNDIALS time integration packages or the KINSOL package, the PCG SUNLinearSolver is bundled with the package library and it is not necessary to link to the library below when using those packages.

Table 1.63: The PCG SUNLinearSolver library, header file, and CMake target

Libraries	`libsundials_sunlinsolpcg.LIB`
Headers	`sunlinsol/sunlinsol_pcg.h`
CMake target	`SUNDIALS::sunlinsolpcg`

1.7.5.12 Scaled, Preconditioned Bi-Conjugate Gradient, Stabilized (SPBCGS)

To use the SPBCGS SUNLinearSolver, include the header file and link to the library given below.

When using SUNDIALS time integration packages or the KINSOL package, the SPBCGS SUNLinearSolver is bundled with the package library and it is not necessary to link to the library below when using those packages.

Table 1.64: The SPBCGS SUNLinearSolver library, header file, and CMake target

Libraries	`libsundials_sunlinsolspbcgs.LIB`
Headers	`sunlinsol/sunlinsol_spbcgs.h`
CMake target	`SUNDIALS::sunlinsolspbcgs`

1.7.5.13 Scaled, Preconditioned, Flexible, Generalized Minimum Residual (SPFGMR)

To use the SPFGMR SUNLinearSolver, include the header file and link to the library given below.

When using SUNDIALS time integration packages or the KINSOL package, the SPFGMR SUNLinearSolver is bundled with the package library and it is not necessary to link to the library below when using those packages.

Table 1.65: The SPFGMR SUNLinearSolver library, header file, and CMake target

Libraries	`libsundials_sunlinsolspfgmr.LIB`
Headers	`sunlinsol/sunlinsol_spfgmr.h`
CMake target	`SUNDIALS::sunlinsolspfgmr`

1.7.5.14 Scaled, Preconditioned, Generalized Minimum Residual (SPGMR)

To use the SPGMR SUNLinearSolver, include the header file and link to the library given below.

When using SUNDIALS time integration packages or the KINSOL package, the SPGMR SUNLinearSolver is bundled with the package library and it is not necessary to link to the library below when using those packages.

Table 1.66: The SPGMR SUNLinearSolver library, header file, and CMake target

Libraries	`libsundials_sunlinsolspgmr.LIB`
Headers	`sunlinsol/sunlinsol_spgmr.h`
CMake target	`SUNDIALS::sunlinsolspgmr`

1.7.5.15 Scaled, Preconditioned, Transpose-Free Quasi-Minimum Residual (SPTFQMR)

To use the SPTFQMR SUNLinearSolver, include the header file and link to the library given below.

When using SUNDIALS time integration packages or the KINSOL package, the SPTFQMR SUNLinearSolver is bundled with the package library and it is not necessary to link to the library below when using those packages.

Table 1.67: The SPTFQMR SUNLinearSolver library, header file, and CMake target

Libraries	`libsundials_sunlinsolsptfqmr.LIB`
Headers	`sunlinsol/sunlinsol_sptfqmr.h`
CMake target	`SUNDIALS::sunlinsolsptfqmr`

1.7.5.16 SuperLU_DIST

To use the SuperLU_DIST SUNLinearSolver, include the header file and link to the library given below.

Table 1.68: The SuperLU_DIST SUNLinearSolver library, header file, and CMake target

Libraries	`libsundials_sunlinsolsuperludist.LIB`
Headers	`sunlinsol/sunlinsol_superludist.h`
CMake target	`SUNDIALS::sunlinsolsuperludist`

1.7.5.17 SuperLU_MT

To use the SuperLU_MT SUNLinearSolver, include the header file and link to the library given below.

Table 1.69: The SuperLU_MT SUNLinearSolver library, header file, and CMake target

Libraries	`libsundials_sunlinsolsuperlumt.LIB`
Headers	`sunlinsol/sunlinsol_superlumt.h`
CMake target	`SUNDIALS::sunlinsolsuperlumt`

1.7.6 Nonlinear Solvers

1.7.6.1 Newton

To use the Newton SUNNonlinearSolver, include the header file and link to the library given below.

When using SUNDIALS time integration packages, the Newton SUNNonlinearSolver is bundled with the package library and it is not necessary to link to the library below when using those packages.

Table 1.70: The Newton SUNNonlinearSolver library, header file, and CMake target

Libraries	`libsundials_sunnonlinsolnewton.LIB`
Headers	`sunnonlinsol/sunnonlinsol_newton.h`
CMake target	`SUNDIALS::sunnonlinsolnewton`

1.7.6.2 Fixed-point

To use the fixed-point SUNNonlinearSolver, include the header file and link to the library given below.

When using SUNDIALS time integration packages, the fixed-point SUNNonlinearSolver is bundled with the package library and it is not necessary to link to the library below when using those packages.

Table 1.71: The Fixed-point SUNNonlinearSolver library, header file, and CMake target

Libraries	`libsundials_sunnonlinsolfixedpoint.LIB`
Headers	`sunnonlinsol/sunnonlinsol_fixedpoint.h`
CMake target	`SUNDIALS::sunnonlinsolfixedpoint`

1.7.6.3 PETSc SNES

To use the PETSc SNES SUNNonlinearSolver, include the header file and link to the library given below.

Table 1.72: The PETSc SNES SUNNonlinearSolver library, header file, and CMake target

Libraries	`libsundials_sunnonlinsolpetscsnes.LIB`
Headers	`sunnonlinsol/sunnonlinsol_petscsnes.h`
CMake target	`SUNDIALS::sunnonlinsolpetscsnes`

1.7.7 Memory Helpers

1.7.7.1 System

When using SUNDIALS time integration packages or the KINSOL package, the system SUNMemoryHelper is bundled with the package library and it is not necessary to link to the library below when using those packages.

Table 1.73: SUNDIALS system memory helper header file

Headers	`sunmemory/sunmemory_system.h`

1.7.7.2 CUDA

To use the CUDA SUNMemoryHelper, include the header file given below when using a CUDA-enabled NVector or SUNMatrix.

Table 1.74: SUNDIALS CUDA memory helper header file

Headers	`sunmemory/sunmemory_cuda.h`

1.7.7.3 HIP

To use the HIP SUNMemoryHelper, include the header file given below when using a HIP-enabled NVector or SUNMatrix.

Table 1.75: SUNDIALS HIP memory helper header file

Headers	`sunmemory/sunmemory_hip.h`

1.7.7.4 SYCL

To use the SYCL SUNMemoryHelper, include the header file given below when using a SYCL-enabled NVector or SUNMatrix.

Table 1.76: SUNDIALS SYCL memory helper header file

Headers	`sunmemory/sunmemory_sycl.h`

1.7.8 Execution Policies

1.7.8.1 CUDA

When using a CUDA-enabled NVector or SUNMatrix, include the header file below to access the CUDA execution policy C++ classes.

Table 1.77: SUNDIALS CUDA execution policies header file

Headers	`sundials/sundials_cuda_policies.hpp`

1.7.8.2 HIP

When using a HIP-enabled NVector or SUNMatrix, include the header file below to access the HIP execution policy C++ classes.

Table 1.78: SUNDIALS HIP execution policies header file

Headers	`sundials/sundials_hip_policies.hpp`

1.7.8.3 SYCL

When using a SYCL-enabled NVector or SUNMatrix, include the header file below to access the SYCL execution policy C++ classes.

Table 1.79: SUNDIALS SYCL execution policies header file

Headers	`sundials/sundials_sycl_policies.hpp`

1.7.9 Adjoint Sensitivity Checkpointing

1.7.9.1 Fixed ASA checkpointing

For fixed-interval adjoint checkpointing, include the header file below:

Table 1.80: SUNDIALS fixed adjoint checkpointing header files

Headers	`sunadjointcheckpointscheme/sunadjointcheckpointscheme_fixed.h`

Index