# VE CMake Toolchain User Guide

Revision 1.2

**Revision History**

| Rev. | Date | Updates / Remarks |
|---|---|---|
| 1.0 | Sep-2021 | The first revision<br>This revision covers VEOS v2.9.1 or later. |
| 1.1 | Sep-2023 | Updating the toolchain files for a specific project<br>This revision covers VEOS v3.2.1 or later. |
| 1.2 | Sep-2024 | Enabling C/C++ Standard Language for Existing/New Project<br>This revision covers VEOS v3.4 or later |

# Table of Contents

# 1. Introduction

CMake is an open-source, cross-platform family of tools designed to build, test and package software. CMake uses a toolchain of utilities to compile, link libraries and create archives, and other tasks to drive the build. The toolchain utilities available are determined by the languages enabled. In normal builds, CMake automatically determines the toolchain for host builds based on system introspection and defaults. In cross-compiling scenarios, a toolchain file may be specified with information about compiler and utility paths.

If cmake is invoked with the command line parameter `-DCMAKE_TOOLCHAIN_FILE=path/to/file`, the file will be loaded early to set values for the compilers.

For SX-Aurora TSUBASA, we have the following requirement:

1. Porting an existing program for VE using CMake
2. Building a new program for VE using CMake

For both the use cases, cmake-ve package is required. This cmake-ve package provides the toolchain configuration file, which will be used for generating Makefile for VE. In case of VE, users on VH Host OS may want to generate the Makefile for VE architecture. Hence, x86 cmake binary running with toolchain file of cmake-ve package would generate Makefile and other configuration files as per VE architecture.

This document explains the usage of VE toolchain configuration file for the two requirements.

This user guide is intended to be used for cmake-ve, with VEOS v3.2.1 or later.

# 2. Installation

## 2.1 Prerequisites

The ‘cmake’ package (x86 based) must be installed.

## 2.2 Installing cmake-ve package

Install cmake-ve.

```
# yum install cmake-ve
```

Upon installation, the tool chain configuration file `toolchainVE.cmake` and `toolchainVE-MPI.cmake` will be placed in `/opt/nec/ve/share/cmake` path.

- The tool chain configuration file – `/opt/nec/ve/share/cmake/toolchainVE.cmake` will be used to port a new program / build an existing program (non-MPI) for VE.
- The tool chain configuration file – `/opt/nec/ve/share/cmake/toolchainVE-MPI.cmake` will be used to port a new program / build an existing program (MPI) for VE.

# 3. Use Cases

## 3.1 Use Case-1: Porting an Existing Program for VE

To port an existing program for VE, follow the below mentioned steps:

```
1. Move to the project directory
$ cd <project_directory_path>

2. To set up the MPI compilation environment, execute the following script
(This execution step is only applicable for MPI programs. For non-MPI
program, skip this step)

$ source /opt/nec/ve/mpi/<version>/bin/necmpivars.sh
Where{version} is the directory name corresponding to the version of NEC MPI
you use.

3. If BLAS and LAPACK library is used in project, then execute the following
script
$ source /opt/nec/ve/nlc/<version>/bin/nlcvars.sh
Where{version} is the directory name corresponding to the version of NEC
LAPACK and BLAS libraries you use.

4. Create a build directory inside the project directory and move into it
$ mkdir build
$ cd build

5. Execute the cmake command inside build directory

For non-MPI projects:
cmake -DCMAKE_TOOLCHAIN_FILE=/opt/nec/ve/share/cmake/toolchainVE.cmake
<project_directory_path>

For MPI projects:
cmake -DCMAKE_TOOLCHAIN_FILE=/opt/nec/ve/share/cmake/toolchainVE-MPI.cmake
<project_directory_path>


6. Finally build the project
$ cmake --build .
OR
$ make
```

## 3.2 Use Case-2: Building a new program for VE

This case is similar to use case-1. The requirement is to build a new program for VE using CMake build system and VE toolchain configuration file

1. Create the new project using CMake (with 'CMakeLists.txt' as an input to CMake build system). Refer the cmake tutorial available at cmake.org
   https://cmake.org/cmake/help/latest/guide/tutorial/index.html.
2. Use the VE toolchain configuration file for building the project to VE. The usage of VE toolchain configuration file is same as Use Case-1.

## 3.3 Use Case-3: Updating the toolchain files for a specific project

This case is similar to use case-1. The requirement is to build a new program for VE using CMake build system and VE toolchain configuration file.

In case user encounter build failure in a specific project, then user needs to update VE toolchain configuration file. We have described the instructions to update VE toolchain configuration file.

```
1.Firstly, copy the VE toolchain configuration file to their home directory.

For non-MPI projects:
$ cp /opt/nec/ve/share/cmake/toolchainVE.cmake   /home/user1/.

For MPI projects:
$ cp /opt/nec/ve/share/cmake/toolchainVE-MPI.cmake /home/user1/.

2.Update VE toolchain configuration file by removing line which contains
"CMAKE_Fortran_COMPILER_ID".

Remove below line: -
  "set( CMAKE_Fortran_COMPILER_ID "NEC" CACHE STRING "Aurora Fortran
compiler ID")"

3.Now, Execute the cmake command with updated VE toolchain configuration
file.

For non-MPI projects:
cmake -DCMAKE_TOOLCHAIN_FILE=/home/user1/toolchainVE.cmake
<project_directory_path>

For MPI projects:
cmake -DCMAKE_TOOLCHAIN_FILE=/home/user1/toolchainVE-MPI.cmake
<project_directory_path>

4.Execute all steps of Use Case-1
```

*Note: The usage of VE toolchain configuration file simplifies the building of a new project to VE. This can also handle the requirement of building the project for both x86_64 or VE architecture. If VE toolchain configuration file is used, then project is built for VE, else it is built for x86_64*

### 3.4 Use Case-4: Enabling C/C++ Standard Language for Existing/New Project for VE.

The requirement is to build a new or existing project for VE with specific language standard such as C/C++ standard.

In case user encounter a build failure in a specific project for VE and that is related to compilation error due to inappropriate C/C++ standard language, then user can enable the appropriate C/C++ standard through CMAKE variables.

For Example: -

To enable C++ Standard flag such as c++17, user can append "-std=c++17" in existing CMAKE variables like "CMAKE_CXX_FLAGS" in user's project i.e. " <user's project path>/CMakeLists.txt". The user needs to set the flags after the "project" command in CMakeLists.txt.

```
cmake_minimum_required(VERSION xx)
project(test)
…
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++17")
…
```

Similarly, user can enable C Standard flag by using "CMAKE_C_FLAGS" in CMakeLists.txt.

## 4. Example

This section highlights the use cases with an example.

The example programs are available on github:
https://github.com/veos-sxarr-NEC/examples/tree/master/cmake

### 4.1 Example of Porting an Existing Program for VE

This example shows the steps to port an existing C program (non-MPI) for VE. The process is same for C++ and Fortran programs

```
1. Move to the project directory
$ cd ~/example/cmake_projects/C/

2. Create a build directory inside the project directory and move into it
$ mkdir build
$ cd build

3. If BLAS and LAPACK library is used in project, then execute the following
script
$ source /opt/nec/ve/nlc/<version>/bin/nlcvars.sh
Where{version} is the directory name corresponding to the version of NEC
LAPACK and BLAS libraries you use.

4. Execute the cmake command inside build directory
cmake -DCMAKE_TOOLCHAIN_FILE=/opt/nec/ve/share/cmake/toolchainVE.cmake ../
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
```

```
-- Check for working C compiler: /opt/nec/ve/bin/ncc - skipped
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /opt/nec/ve/bin/nc++ - skipped
-- Configuring done
-- Generating done
-- Build files have been written to: /home/user1/example/cmake_projects
/C/build

5. Finally build the project
$ cmake --build .
Scanning dependencies of target sample_bin
[ 50%] Building C object CMakeFiles/sample_bin.dir/sample.c.o
[100%] Linking C executable sample_bin
[100%] Built target sample_bin
OR
$ make
Scanning dependencies of target sample_bin
[ 50%] Building C object CMakeFiles/sample_bin.dir/sample.c.o
[100%] Linking C executable sample_bin
[100%] Built target sample_bin

6. Run VE binary
$ /opt/nec/ve/bin/ve_exec ./sample_bin
Hello World
```

This example shows the steps to port an existing C program (MPI) for VE. The process is same for C++ and Fortran programs

```
1. Move to the project directory
$ cd ~/example/MPI_cmake_projects/MPI_C/

2.Setup environment for MPI program
$ source /opt/nec/ve/mpi/2.18.0/bin/necmpivars.sh

3. If BLAS and LAPACK library is used in project, then execute the following
script:
$ source /opt/nec/ve/nlc/<version>/bin/nlcvars.sh
Where{version} is the directory name corresponding to the version of NEC
LAPACK and BLAS libraries you use.

4. Create a build directory inside the project directory and move into it
$ mkdir build
$ cd build

5. Execute the cmake command inside build directory
cmake -DCMAKE_TOOLCHAIN_FILE=/opt/nec/ve/share/cmake/toolchainVE-
MPI.cmake ../
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /opt/nec/ve/mpi/2.18.0/bin/mpicc - skipped
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /opt/nec/ve/mpi/2.18.0/bin/mpic++ -
skipped
-- Found MPI_C: /opt/nec/ve/mpi/2.18.0/bin/mpicc (found version "3.1")
```

```
-- Found MPI_CXX: /opt/nec/ve/mpi/2.18.0/bin/mpic++ (found version "3.1")
-- Found MPI: TRUE (found version "3.1")
-- Configuring done
-- Generating done
-- Build files have been written to:
/home/user1/example/MPI_cmake_projects/MPI_C/build

6. Finally build the project
$ cmake --build .
Scanning dependencies of target test_mpi_bin
[ 50%] Building CXX object CMakeFiles/test_mpi_bin.dir/MPI.c.o
[100%] Linking CXX executable test_mpi_bin
[100%] Built target test_mpi_bin
OR
$ make
Scanning dependencies of target test_mpi_bin
[ 50%] Building CXX object CMakeFiles/test_mpi_bin.dir/MPI.c.o
[100%] Linking CXX executable test_mpi_bin
[100%] Built target test_mpi_bin

7. Run VE binary
$ /opt/nec/ve/bin/mpirun test_mpi_bin
Hello MPI C
Rank ID: 0
Number of processes: 1
Host name: localhost
```

## 4.2 Example of building a new Program for VE

Usage of VE toolchain configuration file for building a new program for VE is same as Example 4.1. Please refer Example 4.1 for more details.