User Guide for CMake and BoolSPLG CUDA Library

Dushan Bikov [*] dusan.bikov@ugd.edu.mk

Iliya Bouyukliev [†] iliyab@math.bas.bg

VERSION 0.3, updated March 13, 2023

Abstract

This guide explain how can automate the whole process of BoolSPLG (Boolean functions and S-box Parallel Library for GPU) CUDA library installation and build the programs examples. Here also is presented use of CMake software on Windows and Linux.

Contents

1 CMake

CMake is free open-source BSD [1] license software, powerful cross-platform system of tools designed for build automation, testing, packaging and installation of software by using compiler-independent manner [2]. It is designed to be used in conjunction with the native build environment such as Make, Qt Creator, Android Studio, Microsoft Visual Studio, and etc. CMake supports directory hierarchies and uses independent simple configuration files called *CMakeLists.txt* placed in each source directory. These configuration files are used to generate native *makefiles* standard build files and workspaces that can be used to compile the source code in the native compiler environment by choice. CMake supports static and dynamic library builds, generates wrappers and build executables. It also supports in-place and out-of-place builds, which means that multiple builds can be derived from a single source tree. Convenient CMake feature is generation of cache file where information about files' locations, libraries, executables, and optional build directives is gathered. The cache file is designed to be used with a graphical editor. The gathered cache information my be changed by the user prior generating build files.

The design of CMake supports applications dependent on several libraries combined by complex directory hierarchies support that allow project to consist multiple toolkits. Each toolkit can contain several directories, and the application depends on the toolkits plus additional code. The CMake can handle various situations, for instance there is situation where executables must be built in order to generate code which is then compiled and linked into a final application. CMake has simple, extensible design and because it is open source it allows expansion with new features.

The CMake is simple to use. The build process is controlled by one or more *CMakeLists.txt* configuration files placed in the project directory and subdirectories. Every configuration file *CMakeLists.txt* consists of one or more commands. The command placed in configuration file have the form "COMMAND(args...)" where COMMAND is the name of the command, and *args* is a list of arguments separate by white-space. CMake sustains many pre-defined commands, and is possible to add custom commands. Additionally, the more advanced users can add custom *makefile* generators for a particular compiler/OS combination.

[*]Faculty of Computer Science, Goce Delchev University, Shtip, Macedonia

[†]Institute of Mathematics and Informatics, Veliko Tarnovo, Bulgaria

2 Install BoolSPLG

BoolSPLG library is implemented as a C++ header library. The minimal CUDA version is 3.x that is required to be able to use BoolSPLG library. It is recommended to use newer than 9.x and above, because some of the used functions are deprecated in the old CUDA versions. There is no need to "build" BoolSPLG library separately. To be able to use BoolSPLG primitives in your code, simply:

1. Clone the repository from *GitHub* to your local computer. Cloning the BoolSPLG-v0.3 library resource folder:

   ```
   $cd "directory"
   $ git clone https://github.com/BoolSPLG/BoolSPLG-v0.3
   ...
   $
   ```

   The newly created BoolSPLG-v0.3 folder in your HOME is the place where is install the BoolSPLG library resources.

   You need to install (copy) only "BoolSPLG" subdirectory from the main BoolSPLG-vx.x directory to the CUDA include directory. CUDA include directory usually is:

   - "/usr/include/" on a Linux (Ubuntu);
   - "C:\CUDA\include\" on a Windows.

   If you are unable to install BoolSPLG to the CUDA include directory, then you can place BoolSPLG somewhere in your home directory.

2. #include the "umbrella" <BoolSPLG/BoolSPLG_v03cuh>header file in your CUDA C++ sources;

3. Compile your program with NVIDIA nvcc CUDA compiler, specifying a -I<path-to- BoolSPLG>include-path flag to reference the location of the BoolSPLG header library.

2.1 Program Examples

BoolSPLG distribution directory contains "Examples" directory with examples (CodeExample, PropertiesGPU, Boolean and S-box) programs. All examples' sub directories contain couple header files and main file with extension ".cu". The main file calls the main function. There are additional header files in the examples directory that contain CPU Boolean and S-box functions that are used for comparison and checking the obtained results. The example directory "ExampleSboxBoolSPLG" apart the main and additional header files contains "sbox" file which is input example S-box. To able to run ExampleSboxBoolSPLG you need to copy "sbox" in program file directory.

There is also separate directory "help and additional header files". There are all additional header files in this directory. Besides these files there is also S-box examples input files whit various S-box sizes.

3 BoolSPLG and CMake

CMake can help to automate the whole process of BoolSPLG library installation. Beside the library installation, CMake also can be use to build the "Examples" programs. The configuration files *CMakeLists.txt* placed in the each source directory are used to generate standard build files on a different platform environment.

The use of CMake latest version is recommended, or at least 3.21. The basic CMake setting up require selecting the configuration file *CMakeLists.txt* and selecting the where to build the binaries. Into the main configuration file *CMakeLists.txt* the required CUDA version is set to 10.0. The necessary version can be changed by changing the argument of the configuration command (CUDA **version** REQUIRED). Keep in mind that CMake not always recognizes the correct GPU architecture, this may also be subject of adjustment. By pressing of button "Configure" from the graphic interface will show it is any configuration problems. If there isn't any configuration problems you may proceed with the generating native *makefiles* standard build files (projects) by pressing the button "Generate".

3.1 BoolSPLG Install and use with CMake on Windows

The native IDE for CUDA applications on Windows is MSVC (Microsoft Visual Studio). CMake is designed to be used as connector with native build environment and the configuration file creates projects in Windows MSVC.

CMake can help to automate the generating the BoolSPLG-vx.x MSVC project. After the basic CMake setting up and before to generate the MSVC project it is necessary to check GPU architecture into CMake graphic interface.

- It is possible CMake do not recognise the correct GPU architecture. This will require to set correct GPU architecture before generating MSVC project by change configuration field CMAKE_CUDA_ARCHITECTURE with correct argument value.

The MSVC project must be run with **administrator** rights so that it can copy library files into CUDA include directory in created "BoolSPLG" directory. Generated BoolSPLG-vx.x MSVC project contains few targets as INSTALL, CodeExample, Properties, Boolean and Sbox. Target INSTALL copies the header library files into CUDA include directory and through the other four targets you could build, compile and execute the examples programs.

3.2 BoolSPLG Install and use with CMake on Linux

On Linux it is needed to generate Makefile for the each target directory. The *Makefiles* can be generated from CMake through the graphical or terminal interface.

General approach when use CMake with graphic interface is the same as in Windows. Here CMake is used to generate *Makefiles* in the predefined directories which are standard build files for the Linux environment. After *Makefiles* generation it is needed to execute the command *"sudo make install"* in terminal. Before executing the command the terminal current working directory needs to be set where is the main *Makefile*. This command will install (copy) library header files into CUDA include directory in created "BoolSPLG" directory. Apart of the installation the command will create four (CodeExample, Properties, Boolean and Sbox) for the examples programs.

The use of the terminal interface implies the use of few basic commands as *mkdir*, *cd* and *cmake*. The command *mkdir* will be use to create "build" directory. The command *cd* is used to change the current working directory. Command *cmake* is a *Makefile* generator. Executing this command in some particular order with additional option that will point the directory of main configuration file *CMakeLists.txt* will generate the *Makefile* files.

***Example** how to use CMake through the terminal.

- Open the terminal and go to the working directory (project source directory);
- Create a build directory in the top source directory: *mkdir build*
- Change the terminal working directory, go inside it: *cd build*
- Run *cmake* command and point to the parent directory: *cmake ..*
- Finally run *cmake* or in our case: *sudo make install*

Notice that *make* and *cmake* are different commands.

The compile and build of the examples programs practically is done with the command *make*. Executable file is generated by using the command *make* on the example target name. To run example executable file first is need to set example working directory into the terminal. After setting up example working directory example program can be run with the command "./target name". If is necessary to debug the project it is much more comfortable to use IDE.

***Note.** There is possible for an error to occur while install library through the terminal. In this case comment the targets in the main configuration file *CMakeLists.txt* and after library installation build the examples separately.

3.2.1 BoolSPLG import into Nsight Eclipse

There is no native IDE for CUDA application on Linux. Eclipse with addition plugin Nsight can be use as integrated environment to edit, build, debug and profile CUDA C applications. One possible way to import existing CUDA project (who uses CMake) into Eclipse Nsight is by following these steps ([3] with suitable modification).

Before you generate *Makefiles*, there is need to set the path to build the binaries. Create directory "build" into main directory of BoolSPLG CMake project and set the path. After generating the *Makefiles* through CMake the next steps are:

1. In Eclipse, go to File->New->CUDA C/C++ Project;
2. Uncheck "Use default location" and select your root directory, BoolSPLG CMake project directory;
3. Set the "Project name:" as the name of your BoolSPLG CMake project;
4. In the Project Type tree select "Executable/Empty Project";
5. Select "CUDA Toolkit on the path" in the Toolchains list;
6. Finish with creating of the project;
7. Before to **build the project** make sure to make last adjustment:

    - Change "Build location" in the project Properties to points to the main *Makefiles* location;

    Example: right click: Project name ->Properties ->Build ->Build location

    In "Makefile generation" uncheck "Generate Makefiles automatically" and set: Build directory: ${workspace_loc:/BOOLSPL_cmake}/**build**

    Note. Depending on the IDE version, setting up this step can be adjusted in the last step of project creation. In some IDE version there is button "Advanced settings" that can be use before to "Finish" with project creation.
8. After building successfully, right click: Project name ->Run As ->Local C/C++ Application, then select which binary you want to execute.

If some of the files didn't show into IDE Project Explorer but they exist into the file system, refresh the project by right click: Project name ->Refresh.

Here is presented general frame for using CMake with Nsight Eclipse development environment. Of course it can customise additional IDE properties option that will make the environment more user friendly.

Acknowledgments

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan X Pascal GPU used for this research.

The development of the library is supported by the Bulgarian National Science Fund under Contract No KP-06-N32/2-2019 and Contract No KP-06-N62/2/13.12.2022.

References

[1] BSD Licenses CMake, Availaible on:
`https://gitlab.kitware.com/cmake/cmake/-/tree/master/Licenses`

[2] CMake official web site, Availaible on:
`https://cmake.org/`

[3] Import existing CUDA project into Nsight web site, Availaible on:
`https://nanxiao.me/en/import-existing-cuda-project-into-nsight/`