# Table of Contents

1. [Contributing to Thrust](#contributing-to-thrust)

1. [CMake Options](#cmake-options)

1. [Development Model](#development-model)

# Contributing to Thrust

Thrust uses Github to manage all open-source development, including bug

tracking, pull requests, and design discussions. This document details how to get

started as a Thrust contributor.

An overview of this process is:

1. [Clone the Thrust repository](#clone-the-thrust-repository)

1. [Setup a fork of Thrust](#setup-a-fork-of-thrust)

1. [Setup your environment](#setup-your-environment)

1. [Create a development branch](#create-a-development-branch)

1. [Local development loop](#local-development-loop)

1. [Push development branch to your fork](#push-development-branch-to-your-fork)

1. [Create pull request](#create-pull-request)

1. [Address feedback and update pull request](#address-feedback-and-update-pull-request)

1. [When your PR is approved...](#when-your-pr-is-approved)

## Clone the Thrust Repository

To get started, clone the main repository to your local computer. Thrust should

be cloned recursively to setup the CUB submodule (required for `CUDA`

acceleration).

```

git clone --recursive https://github.com/thrust/thrust.git

cd thrust

```

## Setup a Fork of Thrust

You'll need a fork of Thrust on Github to create a pull request. To setup your

fork:

1. Create a Github account (if needed)

2. Go to [the Thrust Github page](https://github.com/thrust/thrust)

3. Click "Fork" and follow any prompts that appear.

Once your fork is created, setup a new remote repo in your local Thrust clone:

```

git remote add github-fork git@github.com:<GITHUB\_USERNAME>/thrust.git

```

If you need to modify CUB, too, go to

[the CUB Github page](https://github.com/thrust/cub) and repeat this process.

Create CUB's `github-fork` remote in the `thrust/dependencies/cub` submodule.

## Setup Your Environment

### Git Environment

If you haven't already, this is a good time to tell git who you are. This

information is used to fill out authorship information on your git commits.

```

git config --global user.name "John Doe"

git config --global user.email johndoe@example.com

```

### Configure CMake builds

Thrust uses [CMake](https://www.cmake.org) for its developer build system. To

configure, build, and test your checkout of Thrust:

```

# Create build directory:

mkdir build

cd build

# Configure -- use one of the following:

cmake .. # Command line interface.

ccmake .. # ncurses GUI (Linux only)

cmake-gui # Graphical UI, set source/build directories in the app

# Build:

cmake --build . -j <num jobs> # invokes make (or ninja, etc)

# Run tests and examples:

ctest

```

See [CMake Options](#cmake-options) for details on customizing the build.

## Create a Development Branch

All work should be done in a development branch (also called a "topic branch")

and not directly in the `master` branch. This makes it easier to manage multiple

in-progress patches at once, and provides a descriptive label for your patch

as it passes through the review system.

To create a new branch based on the current `master`:

```

# Checkout local master branch:

cd /path/to/thrust/sources

git checkout master

# Sync local master branch with github:

git pull

# Create a new branch named `my\_descriptive\_branch\_name` based on master:

git checkout -b my\_descriptive\_branch\_name

# Verify that the branch has been created and is currently checked out:

git branch

```

Thrust branch names should follow a particular pattern:

- For new features, name the branch `feature/<name>`

- For bugfixes associated with a github issue, use `bug/github/<bug-description>-<bug-id>`

- Internal nvidia and gitlab bugs should use `nvidia` or `gitlab` in place of

`github`.

If you plan to work on CUB as part of your patch, repeat this process in the

`thrust/dependencies/cub` submodule.

## Local Development Loop

### Edit, Build, Test, Repeat

Once the topic branch is created, you're all set to start working on Thrust

code. Make some changes, then build and test them:

```

# Implement changes:

cd /path/to/thrust/sources

emacs thrust/some\_file.h # or whatever editor you prefer

# Create / update a unit test for your changes:

emacs testing/some\_test.cu

# Check that everything builds and tests pass:

cd /path/to/thrust/build/directory

cmake --build . -j <num jobs>

ctest

```

### Creating a Commit

Once you're satisfied with your patch, commit your changes:

#### Thrust-only Changes

```

# Manually add changed files and create a commit:

cd /path/to/thrust

git add thrust/some\_file.h

git add testing/some\_test.cu

git commit

# Or, if possible, use git-gui to review your changes while building your patch:

git gui

```

#### Thrust and CUB Changes

```

# Create CUB patch first:

cd /path/to/thrust/dependencies/cub

# Manually add changed files and create a commit:

git add cub/some\_file.cuh

git commit

# Create Thrust patch, including submodule update:

cd /path/to/thrust/

git add dependencies/cub # Updates submodule info

git add thrust/some\_file.h

git add testing/some\_test.cu

git commit

# Or, if possible, use git-gui to review your changes while building your patch:

cd /path/to/thrust/dependencies/cub

git gui

cd /path/to/thrust

git gui # Include dependencies/cub as part of your commit

```

#### Writing a Commit Message

Your commit message will communicate the purpose and rationale behind your

patch to other developers, and will be used to populate the initial description

of your Github pull request.

When writing a commit message, the following standard format should be used,

since tools in the git ecosystem are designed to parse this correctly:

```

First line of commit message is a short summary (<80 char)

<Second line left blank>

Detailed description of change begins on third line. This portion can

span multiple lines, try to manually wrap them at something reasonable.

Blank lines can be used to separate multiple paragraphs in the description.

If your patch is associated with another pull request or issue in the main

Thrust repository, you should reference it with a `#` symbol, e.g.

#1023 for issue 1023.

For issues / pull requests in a different github repo, reference them using

the full syntax, e.g. thrust/cub#4 for issue 4 in the thrust/cub repo.

Markdown is recommended for formatting more detailed messages, as these will

be nicely rendered on Github, etc.

```

## Push Development Branch to your Fork

Once you've committed your changes to a local development branch, it's time to

push them to your fork:

```

cd /path/to/thrust/checkout

git checkout my\_descriptive\_branch\_name # if not already checked out

git push --set-upstream github-fork my\_descriptive\_branch\_name

```

`--set-upstream github-fork` tells git that future pushes/pulls on this branch

should target your `github-fork` remote by default.

If have CUB changes to commit as part of your patch, repeat this process in the

`thrust/dependencies/cub` submodule.

## Create Pull Request

To create a pull request for your freshly pushed branch, open your github fork

in a browser by going to `https://www.github.com/<GITHUB\_USERNAME>/thrust`. A

prompt may automatically appear asking you to create a pull request if you've

recently pushed a branch.

If there's no prompt, go to "Code" > "Branches" and click the appropriate

"New pull request" button for your branch.

If you would like a specific developer to review your patch, feel free to

request them as a reviewer at this time.

The Thrust team will review your patch, test it on NVIDIA's internal CI, and

provide feedback.

If have CUB changes to commit as part of your patch, repeat this process with

your CUB branch and fork.

## Address Feedback and Update Pull Request

If the reviewers request changes to your patch, use the following process to

update the pull request:

```

# Make changes:

cd /path/to/thrust/sources

git checkout my\_descriptive\_branch\_name

emacs thrust/some\_file.h

emacs testing/some\_test.cu

# Build + test

cd /path/to/thrust/build/directory

cmake --build . -j <num jobs>

ctest

# Amend commit:

cd /path/to/thrust/sources

git add thrust/some\_file.h

git add testing/some\_test.cu

git commit --amend

# Or

git gui # Check the "Amend Last Commit" box

# Update the branch on your fork:

git push -f

```

At this point, the pull request should show your recent changes.

If have CUB changes to commit as part of your patch, repeat this process in the

`thrust/dependencies/cub` submodule, and be sure to include any CUB submodule

updates as part of your commit.

## When Your PR is Approved

Once your pull request is approved by the Thrust team, no further action is

needed from you. We will handle integrating it since we must coordinate changes

to `master` with NVIDIA's internal perforce repository.

# CMake Options

A Thrust build is configured using CMake options. These may be passed to CMake

using

```

cmake -D<option\_name>=<value> /path/to/thrust/sources

```

or configured interactively with the `ccmake` or `cmake-gui` interfaces.

Thrust supports two build modes. By default, a single configuration is built

that targets a specific host system, device system, and C++ dialect.

When `THRUST\_ENABLE\_MULTICONFIG` is `ON`, multiple configurations

targeting a variety of systems and dialects are generated.

The CMake options are divided into these categories:

1. [Generic CMake Options](#generic-cmake-options): Options applicable to all

Thrust builds.

1. [Single Config CMake Options](#single-config-cmake-options) Options

applicable only when `THRUST\_ENABLE\_MULTICONFIG` is disabled.

1. [Multi Config CMake Options](#multi-config-cmake-options) Options applicable

only when `THRUST\_ENABLE\_MULTICONFIG` is enabled.

1. [CUDA Specific CMake Options](#cuda-specific-cmake-options) Options that

control CUDA compilation. Only available when one or more configurations

targets the CUDA system.

1. [TBB Specific CMake Options](#tbb-specific-cmake-options) Options that

control TBB compilation. Only available when one or more configurations

targets the TBB system.

## Generic CMake Options

- `CMAKE\_BUILD\_TYPE={Release, Debug, RelWithDebInfo, MinSizeRel}`

- Standard CMake build option. Default: `RelWithDebInfo`

- `THRUST\_ENABLE\_HEADER\_TESTING={ON, OFF}`

- Whether to test compile public headers. Default is `ON`.

- `THRUST\_ENABLE\_TESTING={ON, OFF}`

- Whether to build unit tests. Default is `ON`.

- `THRUST\_ENABLE\_EXAMPLES={ON, OFF}`

- Whether to build examples. Default is `ON`.

- `THRUST\_ENABLE\_MULTICONFIG={ON, OFF}`

- Toggles single-config and multi-config modes. Default is `OFF` (single config).

- `THRUST\_ENABLE\_EXAMPLE\_FILECHECK={ON, OFF}`

- Enable validation of example outputs using the LLVM FileCheck utility.

Default is `OFF`.

## Single Config CMake Options

- `THRUST\_HOST\_SYSTEM={CPP, TBB, OMP}`

- Selects the host system. Default: `CPP`

- `THRUST\_DEVICE\_SYSTEM={CUDA, TBB, OMP, CPP}`

- Selects the device system. Default: `CUDA`

- `THRUST\_CPP\_DIALECT={11, 14, 17}`

- Selects the C++ standard dialect to use. Default is `14` (C++14).

## Multi Config CMake Options

- `THRUST\_MULTICONFIG\_ENABLE\_DIALECT\_CPPXX={ON, OFF}`

- Toggle whether a specific C++ dialect will be targeted.

- Possible values of `XX` are `{11, 14, 17}`.

- By default, only C++14 is enabled.

- `THRUST\_MULTICONFIG\_ENABLE\_SYSTEM\_XXXX={ON, OFF}`

- Toggle whether a specific system will be targeted.

- Possible values of `XXXX` are `{CPP, CUDA, TBB, OMP}`

- By default, only `CPP` and `CUDA` are enabled.

- `THRUST\_MULTICONFIG\_WORKLOAD={SMALL, MEDIUM, LARGE, FULL}`

- Restricts the host/device combinations that will be targeted.

- By default, the `SMALL` workload is used.

- The full cross product of `host x device` systems results in 12

configurations, some of which are more important than others.

This option can be used to prune some of the less important ones.

- `SMALL`: (3 configs) Minimal coverage and validation of each device system against the `CPP` host.

- `MEDIUM`: (6 configs) Cheap extended coverage.

- `LARGE`: (8 configs) Expensive extended coverage. Includes all useful build configurations.

- `FULL`: (12 configs) The complete cross product of all possible build configurations.

| Config | Workloads | Value | Expense | Note |

|----------|-----------|------------|-----------|------------------------------|

| CPP/CUDA | `F L M S` | Essential | Expensive | Validates CUDA against CPP |

| CPP/OMP | `F L M S` | Essential | Cheap | Validates OMP against CPP |

| CPP/TBB | `F L M S` | Essential | Cheap | Validates TBB against CPP |

| CPP/CPP | `F L M ` | Important | Cheap | Tests CPP as device |

| OMP/OMP | `F L M ` | Important | Cheap | Tests OMP as host |

| TBB/TBB | `F L M ` | Important | Cheap | Tests TBB as host |

| TBB/CUDA | `F L ` | Important | Expensive | Validates TBB/CUDA interop |

| OMP/CUDA | `F L ` | Important | Expensive | Validates OMP/CUDA interop |

| TBB/OMP | `F ` | Not useful | Cheap | Mixes CPU-parallel systems |

| OMP/TBB | `F ` | Not useful | Cheap | Mixes CPU-parallel systems |

| TBB/CPP | `F ` | Not Useful | Cheap | Parallel host, serial device |

| OMP/CPP | `F ` | Not Useful | Cheap | Parallel host, serial device |

## CUDA Specific CMake Options

- `THRUST\_INCLUDE\_CUB\_CMAKE={ON, OFF}`

- If enabled, the CUB project will be built as part of Thrust. Default is

`OFF`.

- This adds CUB tests, etc. Useful for working on both CUB and Thrust

simultaneously.

- CUB configurations will be generated for each C++ dialect targeted by

the current Thrust build.

- `THRUST\_ENABLE\_COMPUTE\_XX={ON, OFF}`

- Controls the targeted CUDA architecture(s)

- Multiple options may be selected when using NVCC as the CUDA compiler.

- Valid values of `XX` are:

`{35, 37, 50, 52, 53, 60, 61, 62, 70, 72, 75, 80}`

- Default value depends on `THRUST\_DISABLE\_ARCH\_BY\_DEFAULT`:

- `THRUST\_ENABLE\_COMPUTE\_FUTURE={ON, OFF}`

- If enabled, CUDA objects will target the most recent virtual architecture

in addition to the real architectures specified by the

`THRUST\_ENABLE\_COMPUTE\_XX` options.

- Default value depends on `THRUST\_DISABLE\_ARCH\_BY\_DEFAULT`:

- `THRUST\_DISABLE\_ARCH\_BY\_DEFAULT={ON, OFF}`

- When `ON`, all `THRUST\_ENABLE\_COMPUTE\_\*` options are initially `OFF`.

- Default: `OFF` (meaning all architectures are enabled by default)

- `THRUST\_ENABLE\_TESTS\_WITH\_RDC={ON, OFF}`

- Whether to enable Relocatable Device Code when building tests.

Default is `OFF`.

- `THRUST\_ENABLE\_EXAMPLES\_WITH\_RDC={ON, OFF}`

- Whether to enable Relocatable Device Code when building examples.

Default is `OFF`.

## TBB Specific CMake Options

- `THRUST\_TBB\_ROOT=<path to tbb root>`

- When the TBB system is requested, set this to the root of the TBB installation

(e.g. the location of `lib/`, `bin/` and `include/` for the TBB libraries).

# Development Model

The following is a description of the basic development process that Thrust follows. This is a living

document that will evolve as our process evolves.

Thrust is distributed in three ways:

\* On GitHub.

\* In the NVIDIA HPC SDK.

\* In the CUDA Toolkit.

## Trunk Based Development

Thrust uses [trunk based development](https://trunkbaseddevelopment.com). There is a single long-lived

branch called `master`. Engineers may create branches for feature development. Such branches always

merge into `master`. There are no release branches. Releases are produced by taking a snapshot of

`master` ("snapping"). After a release has been snapped from `master`, it will never be changed.

## Repositories

As Thrust is developed both on GitHub and internally at NVIDIA, there are three main places where code lives:

\* The Source of Truth, the [public Thrust repository](https://github.com/thrust/thrust), referred to as

`github` later in this document.

\* An internal GitLab repository, referred to as `gitlab` later in this document.

\* An internal Perforce repository, referred to as `perforce` later in this document.

## Versioning

Thrust has its own versioning system for releases, independent of the versioning scheme of the NVIDIA

HPC SDK or the CUDA Toolkit.

Today, Thrust version numbers have a specific [semantic meaning](https://semver.org/).

Releases prior to 1.10.0 largely, but not strictly, followed these semantic meanings.

The version number for a Thrust release uses the following format: `MMM.mmm.ss-ppp`, where:

\* `THRUST\_VERSION\_MAJOR`/`MMM`: Major version, up to 3 decimal digits. It is incremented

when the fundamental nature of the library evolves, leading to widespread changes across the

entire library interface with no guarantee of API, ABI, or semantic compatibility with former

versions.

\* `THRUST\_VERSION\_MINOR`/`mmm`: Minor version, up to 3 decimal digits. It is incremented when

breaking API, ABI, or semantic changes are made.

\* `THRUST\_VERSION\_SUBMINOR`/`ss`: Subminor version, up to 2 decimal digits. It is incremented

when notable new features or bug fixes or features that are API, ABI, and semantic backwards

compatible are added.

\* `THRUST\_PATCH\_NUMBER`/`ppp`: Patch number, up to 3 decimal digits. It is incremented if any

change in the repo whatsoever is made and no other version component has been incremented.

The `<thrust/version.h>` header defines `THRUST\_\*` macros for all of the version components mentioned

above. Additionally, a `THRUST\_VERSION` macro is defined, which is an integer literal containing all

of the version components except for `THRUST\_PATCH\_NUMBER`.

## Branches and Tags

The following tag names are used in the Thrust project:

\* `github/nvhpc-X.Y`: the tag that directly corresponds to what has been shipped in the NVIDIA HPC SDK release X.Y.

\* `github/cuda-X.Y`: the tag that directly corresponds to what has been shipped in the CUDA Toolkit release X.Y.

\* `github/A.B.C`: the tag that directly corresponds to a Thrust version A.B.C.

The following branch names are used in the Thrust project:

\* `github/master`: the Source of Truth development branch of Thrust.

\* `github/old-master`: the old Source of Truth branch, before unification of public and internal repositories.

\* `github/feature/<name>`: feature branch for a feature under development.

\* `github/bug/<bug-system>/<bug-description>-<bug-id>`: bug fix branch, where `bug-system` is `github` or `nvidia`.

\* `gitlab/master`: mirror of `github/master`.

\* `perforce/private`: mirrored `github/master`, plus files necessary for internal NVIDIA testing systems.

On the rare occasion that we cannot do work in the open, for example when developing a change specific to an

unreleased product, these branches may exist on `gitlab` instead of `github`. By default, everything should be

in the open on `github` unless there is a strong motivation for it to not be open.