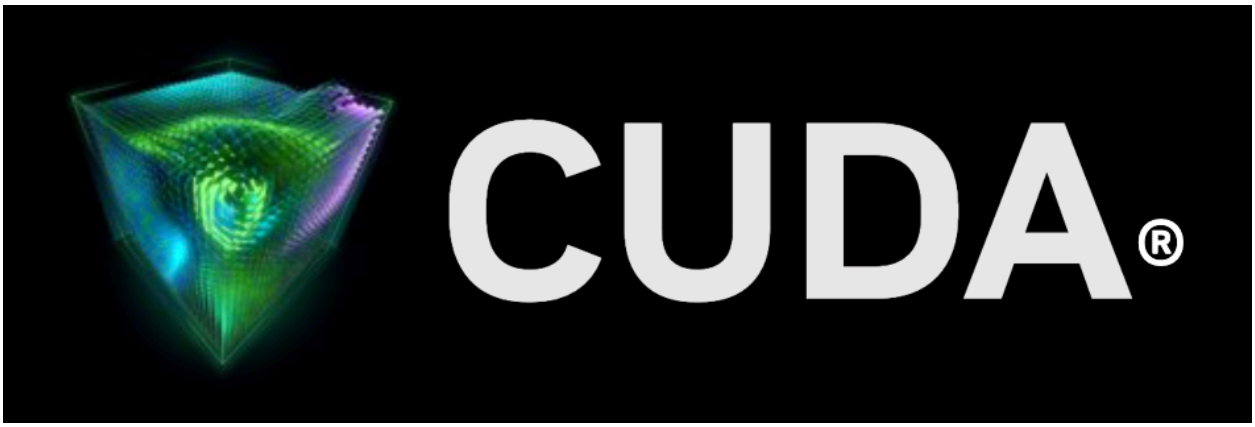

CUDA Compile Time Advisor

Release 13.4

NVIDIA Corporation

Sep 09, 2026

Contents

1 Getting Started **3**
- 1.1 Installation . . . 3
- 1.2 Command-line Options . . . 3
  - 1.2.1 `--trace-file-path` (`-path`) . . . 3
  - 1.2.2 `--thread-number` (`-thread`) . . . 3
  - 1.2.3 `--breakdown-advisor-entries` (`-breakdown-advisor-entries`) . . . 3
  - 1.2.4 `--header-advisor-entries` (`-header-advisor-entries`) . . . 4
  - 1.2.5 `--template-advisor-entries` (`-template-advisor-entries`) . . . 4
  - 1.2.6 `--max-hash-table-size` (`-size`) . . . 4
  - 1.2.7 `--verbose` (`-v`) . . . 4
  - 1.2.8 `--version` (`-V`) . . . 4
  - 1.2.9 `--options-file` (`-optf`) . . . 4
- 1.3 Usage . . . 5

2 Notices **11**
- 2.1 Notice . . . 11
- 2.2 OpenCL . . . 12
- 2.3 Trademarks . . . 12

Compile Time Advisor

The User guide for the Compile Time Advisor (`ctadvisor`).

Compile Time Advisor (`ctadvisor`) is a tool that helps users analyze the compilation time of their CUDA C++ code and provides suggestions to reduce it.

Currently, `ctadvisor` provides five different types of advice:

Expensive Templates Advice:
Identifies template functions/classes that took the longest to instantiate, how many times they were instantiated by which files, how many sub-templates were instantiated, and the depth of the sub-template tree.

Expensive Headers Advice:
Identifies header files that took the longest to include and breaks down the time spent on processing each sub-header file.

Parallelizable Compilation Advice:
Checks if source files were compiled for multiple architectures (multiple `-gencode` targets) and the `--threads` flag was unused. The build can then benefit from using `--threads` to run compilation for each architecture in parallel.

Split Compilation Advice:
Identifies source files with high optimization time that could benefit from the `--split-compile` flag to run optimizations in parallel.

Compile Time Breakdown Advice:
Breaks down the compile time of each translation unit into various compilation steps, lists significant source files that took the largest compilation time, and helps identify which stages of compilation are a bottleneck.

Chapter 1. Getting Started

1.1. Installation

`ctadvisor` is part of the CUDA Toolkit release. It is a standalone tool which is installed automatically with the CUDA Toolkit.

1.2. Command-line Options

1.2.1. `--trace-file-path (-path)`

Specify the location of trace file(s). The path can be one trace file or a directory recursively containing all trace files. This is a required option.

1.2.2. `--thread-number (-thread)`

Specify the maximum number of threads that `ctadvisor` uses to process trace file concurrently. Default value: 8.

1.2.3. `--breakdown-advisor-entries (-breakdown-advisor-entries)`

Specify the maximum number of entries that compile time breakdown advisor is going to show. Default value: 20.

1.2.4. --header-advisor-entries (-header-advisor-entries)

Specify the maximum number of entries that expensive headers advisor is going to show. Default value: 20.

1.2.5. --template-advisor-entries (-template-advisor-entries)

Specify the maximum number of entries that expensive templates advisor is going to show. Default value: 20.

1.2.6. --max-hash-table-size (-size)

Specify the maximum size of the hash table that stores the index of long template names. The template names are stored on disk. And a hash table is used to store the index of template names. Increasing this number will reduce analysis time but increase memory usage. Default value: 100000000.

1.2.7. --verbose (-v)

Enable verbose mode. `ctadvisor` outputs full specialized template names in the analysis of expensive template advisor and more entries in analysis.

1.2.8. --version (-V)

Print version information on this tool.

1.2.9. --options-file (-optf)

Include command line options from specified file.

1.3. Usage

`ctadvisor` accepts a trace file or a directory recursively containing all trace files and then analyzes them. To demonstrate the usage of `ctadvisor`, we will use a simple example here:

```
template <typename T>
class C {
public:
    template <typename S>
    int t_class_fn() {
        return 0;
    }
};

template <typename T>
__global__ void saxpy_kern(int n, T a, T *x, T *y) {
    int t_id = blockIdx.x * blockDim.x + threadIdx.x;
    int stride = blockDim.x * gridDim.x;

    for (int i = t_id; i < n; i+= stride) {
        y[i] = a * x[i] + y[i];
    }
}

template <typename T>
int t_fn() {
    return 0;
}

int main() {
    C<int> c;
    c.t_class_fn<float>();
    t_fn<int>();
    int n = 1UL << 25;
    float *x, *y, alpha=2.0;
    cudaMalloc(&x, n * sizeof(float));
    cudaMalloc(&y, n * sizeof(float));
    saxpy_kern<float><<<32, 1024>>>(n, alpha, x, y);
    cudaDeviceSynchronize();
    cudaFree(x);
    cudaFree(y);
}
```

First, to get the trace file, run nvcc compilation command with `--fdevice-time-trace` flag.

```
nvcc --fdevice-time-trace=- -o saxpy saxpy.cu
```

A `saxpy.json` file is created in the current directory. To analyze the trace file, run `ctadvisor` with the trace file as the argument.

```
ctadvisor -path saxpy.json
```

Here is the output of `ctadvisor`:

```
Trace file loading is complete
************************************************
```

(continues on next page)

(continued from previous page)

```
Start expensive template advisor
No advice.
************************************************
Start expensive headers advisor
Top header files by include processing time:
 [0.05s]: (1 times in 1 files) /usr/local/cuda-13.4/bin/../targets/x86_64-linux/
↪include/cuda_runtime.h
    Processing this header itself takes 0.00s (1.65%)
    includes /usr/local/cuda-13.4/bin/../targets/x86_64-linux/include/crt/common_
↪functions.h (1x) taking 0.03s (58.73%)
    includes /usr/local/cuda-13.4/bin/../targets/x86_64-linux/include/crt/device_
↪functions.h (1x) taking 0.01s (22.84%)
    includes /usr/local/cuda-13.4/bin/../targets/x86_64-linux/include/channel_
↪descriptor.h (1x) taking 0.00s (10.00%)
    includes /usr/local/cuda-13.4/bin/../targets/x86_64-linux/include/builtin_types.h
↪(1x) taking 0.00s (4.44%)
    includes /usr/local/cuda-13.4/bin/../targets/x86_64-linux/include/vector_
↪functions.h (1x) taking 0.00s (1.48%)
    includes /usr/include/c++/12/utility (1x) taking 0.00s (0.58%)
    includes /usr/local/cuda-13.4/bin/../targets/x86_64-linux/include/driver_
↪functions.h (1x) taking 0.00s (0.16%)
    includes /usr/local/cuda-13.4/bin/../targets/x86_64-linux/include/library_types.h
↪(1x) taking 0.00s (0.04%)
    includes /usr/local/cuda-13.4/bin/../targets/x86_64-linux/include/crt/host_config.
↪h (1x) taking 0.00s (0.04%)
    includes /usr/local/cuda-13.4/bin/../targets/x86_64-linux/include/device_launch_
↪parameters.h (1x) taking 0.00s (0.02%)
    includes /usr/local/cuda-13.4/bin/../targets/x86_64-linux/include/crt/host_
↪defines.h (1x) taking 0.00s (0.00%)
************************************************
Start compile time breakdown advisor
1 source files analyzed:
  Gross total compile time: 0.44s
    Gross total host compile time: 0.29s
    Gross total device compile time: 0.15s (frontend: 0.13s, NVVM: 0.00s, ptxas: 0.
↪00s, linker: 0.00s)
Top source files by compile time:
saxpy.cu
    compiled with target architectures: compute_75, threads: 1, split compile:
↪disabled
    Total compile time: net 0.48s, gross 0.44s
    Host compile time: 0.29s    Device compile time: 0.15s (frontend: 0.13s, NVVM: 0.
↪00s, ptxas: 0.00s, linker: 0.00s)
************************************************
Start parallelizable compilation advisor
************************************************
Start split compile advisor
No advice.
```

Since the example is relatively simple, `ctadvisor` does not find any templates that took the significant compilation time. Verbose mode can be enabled by using `-v` to get more detailed information and suggestions.

```
ctadvisor -path saxpy.json -v
```

`--breakdown-advisor-entries`, `--header-advisor-entries` and `--template-advisor-entries` can be used to limit the number of entries that each advisor

is going to show in verbose mode.

```
ctadvisor -path saxpy.json -v --breakdown-advisor-entries 1 --header-advisor-entries
↪1 --template-advisor-entries 1
```

Here is the output of `ctadvisor`:

```
Trace file loading is complete
************************************************
Start expensive template advisor
Top template functions/classes by instantiation time:
  [0.00s]: (2 times in 2 files) std::__detail::__hyperg
    [0.00s]: (1 times in 1 files) std::__detail::__hyperg<float>(float, float, float,
↪float)
      [0.00s]: (1x) saxpy.cu, recursively instantiates 12 templates with max depth 5
    (1 more...)
Try 'extern'-izing templates that are repeatedly instantiated (may impact
↪performance).
Try simplifying template logic for instantiations with deep recursive depth.
************************************************
Start expensive headers advisor
Top header files by include processing time:
 [0.05s]: (1 times in 1 files) /usr/local/cuda-13.4/bin/../targets/x86_64-linux/
↪include/cuda_runtime.h
    Processing this header itself takes 0.00s (1.65%)
    includes /usr/local/cuda-13.4/bin/../targets/x86_64-linux/include/crt/common_
↪functions.h (1x) taking 0.03s (58.73%)
    includes /usr/local/cuda-13.4/bin/../targets/x86_64-linux/include/crt/device_
↪functions.h (1x) taking 0.01s (22.84%)
    includes /usr/local/cuda-13.4/bin/../targets/x86_64-linux/include/channel_
↪descriptor.h (1x) taking 0.00s (10.00%)
    includes /usr/local/cuda-13.4/bin/../targets/x86_64-linux/include/builtin_types.h
↪(1x) taking 0.00s (4.44%)
    includes /usr/local/cuda-13.4/bin/../targets/x86_64-linux/include/vector_
↪functions.h (1x) taking 0.00s (1.48%)
    includes /usr/include/c++/12/utility (1x) taking 0.00s (0.58%)
    includes /usr/local/cuda-13.4/bin/../targets/x86_64-linux/include/driver_
↪functions.h (1x) taking 0.00s (0.16%)
    includes /usr/local/cuda-13.4/bin/../targets/x86_64-linux/include/library_types.h
↪(1x) taking 0.00s (0.04%)
    includes /usr/local/cuda-13.4/bin/../targets/x86_64-linux/include/crt/host_config.
↪h (1x) taking 0.00s (0.04%)
    includes /usr/local/cuda-13.4/bin/../targets/x86_64-linux/include/device_launch_
↪parameters.h (1x) taking 0.00s (0.02%)
    includes /usr/local/cuda-13.4/bin/../targets/x86_64-linux/include/crt/host_
↪defines.h (1x) taking 0.00s (0.00%)
************************************************
Start compile time breakdown advisor
1 source files analyzed:
  Gross total compile time: 0.44s
    Gross total host compile time: 0.29s
    Gross total device compile time: 0.15s (frontend: 0.13s, NVVM: 0.00s, ptxas: 0.
↪00s, linker: 0.00s)
Top source files by compile time:
saxpy.cu
    compiled with target architectures: compute_75, threads: 1, split compile:
↪disabled
    Total compile time: net 0.48s, gross 0.44s
```

(continues on next page)

(continued from previous page)

```
    Host compile time: 0.29s    Device compile time: 0.15s (frontend: 0.13s, NVVM: 0.
↪00s, ptxas: 0.00s, linker: 0.00s)
*******************************************
Start parallelizable compilation advisor
*******************************************
Start split compile advisor
No advice.
```

nvrtc compilation can also generate trace files with `--fdevice-time-trace`. For example:

```
#include <nvrtc.h>
#include <iostream>

#define NVRTC_SAFE_CALL(x)                                        \
    do {                                                          \
        nvrtcResult result = x;                                   \
        if (result != NVRTC_SUCCESS) {                            \
        std::cerr << "\nerror: " #x " failed with error "         \
                    << nvrtcGetErrorString(result) << '\n';       \
        exit(1);                                                  \
    }                                                             \
} while(0)

const char *saxpy = "                                             \n\
extern       \"C\" __global__                                     \n\
void saxpy(float a, float *x, float *y, float *out, size_t n)     \n\
{                                                                 \n\
    size_t tid = blockIdx.x * blockDim.x + threadIdx.x;           \n\
    if (tid < n) {                                                \n\
        out[tid] = a * x[tid] + y[tid];                           \n\
    }                                                             \n\
}                                                                 \n";

int main()
{
    // Create an instance of nvrtcProgram with the SAXPY code string.
    nvrtcProgram prog;
    NVRTC_SAFE_CALL(
        nvrtcCreateProgram(&prog,         // prog
                           saxpy,         // buffer
                           "saxpy.cu",    // name
                           0,             // numHeaders
                           NULL,          // headers
                           NULL));        // includeNames
    // Compile the program with fmad disabled.
    // Note: Can specify GPU target architecture explicitly with '-arch' flag.
    const char *opts[] = {"--fmad=false", "--fdevice-time-trace=saxpy"};
    nvrtcResult compileResult = nvrtcCompileProgram(prog,  // prog
                                                    2,     // numOptions
                                                    opts); // options

    // Destroy the program.
    NVRTC_SAFE_CALL(nvrtcDestroyProgram(&prog));
    // Load the generated PTX and get a handle to the SAXPY kernel.

    return 0;
}
```

Assuming the environment variable CUDA_PATH points to the CUDA Toolkit installation directory, build this example as:

```
g++ saxpy.cpp -o saxpy \
    -I $CUDA_PATH/include \
    -L $CUDA_PATH/lib64 \
    -lnvrtc -Wl,-rpath,$CUDA_PATH/lib64
```

And run this example as:

```
./saxpy
```

A saxpy.json will be generated in the current directory.

Chapter 2. Notices

2.1. Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation ("NVIDIA") makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer's own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or

services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

2.2. OpenCL

OpenCL is a trademark of Apple Inc. used under license to the Khronos Group Inc.

2.3. Trademarks

NVIDIA and the NVIDIA logo are trademarks or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

©2025-2026, NVIDIA Corporation & affiliates. All rights reserved