Tensile Documentation

Release 4.42.0

Advanced Micro Devices, Inc.

Dec 05, 2024

INSTALL

Tensile is a tool for creating a benchmark-driven backend library for General Matrix-Matrix Multiplications (GEMMs), GEMM-like problems such as batched GEMM, N-dimensional tensor contractions, and anything else that multiplies two multidimensional objects together on an AMD GPU.

The code is open source and hosted at https://github.com/ROCm/Tensile

Install

- *Installation*

Conceptual

- *Introduction*
- *Solution selection catalogs*

Reference

- *Environment variables*
- *API reference*
- *CLI reference*

Contribution

- *Programmer's guide*
- *Contribution guidelines*

To contribute to the documentation, refer to Contributing to ROCm.

You can find licensing information on the Licensing page.

CHAPTER

ONE

INSTALLATION

1.1 Install ROCm

To begin, install ROCm for your platform. For installation instructions, refer to the Linux or Windows installation guide.

> **Tip**
>
> If using Bash, we recommend you to set `PATH=/opt/rocm/bin/:$PATH` in your `~/.bashrc` and refresh your shell using `source ~/.bashrc`. Alternatively, export the path for your current shell session only, using `export PATH=/opt/rocm/bin/:$PATH`.

1.2 Install OS dependencies

> **Note**
>
> The steps below are for Ubuntu. For other distributions, use the appropriate package manager.

1. Install dependencies:

```
apt-get install libyaml python3-yaml \
    libomp-dev libboost-program-options-dev libboost-filesystem-dev
```

2. Install one of the following, depending on your preferred Tensile data format. If both are installed, `msgpack` is preferred:

```
apt-get install libmsgpack-dev    # If using the msgpack backend

# OR

apt-get install libtinfo-dev      # If using the YAML backend
```

3. Install build tools. For additional installation methods for the latest versions of CMake, see the CMake installation page.

```
apt-get install build-essential cmake
```

1.3 Install Tensile from source

To install Tensile from source, it is recommended to create a virtual environment first:

```
python3 -m venv .venv
source .venv/bin/activate
```

Then, you can install Tensile using pip or git.

1.3.1 Option 1: Install with pip

```
pip3 install git+https://github.com/ROCmSoftwarePlatform/Tensile.git@develop
```

1.3.2 Option 2: Install with git

```
git clone git@github.com:ROCm/Tensile.git && cd Tensile
pip3 install .
```

You can now run Tensile's Python applications.

1.4 Running benchmark

To run a benchmark, pass a tuning config to the `Tensile` program located in `Tensile/bin`.

For demonstration purposes, we use the sample tuning file available in `Tensile/Configs/rocblas_sgemm_example.yaml`. The sample tuning file allows you to specify the target architecture for which the benchmark will generate a library. To find your device architecture, run:

```
rocminfo | grep gfx
```

Specify the device architecture in the sample tuning file using `ArchitectureName:`. Based on the device architecture, use `ArchitectureName: "gfx90a"` or `ArchitectureName: "gfx1030"`.

You can now run benchmarks using Tensile. From the top-level directory, run:

```
mkdir build && cd build
../Tensile/bin/Tensile ../Tensile/Configs/rocblas_sgemm_example.yaml ./
```

After the benchmark completes, Tensile creates the following directories:

- **0_Build**: Contains a client executable. Use this to launch Tensile from a library viewpoint.
- **1_BenchmarkProblems**: Contains all the problem descriptions and executables generated during benchmarking. Use the `run.sh` script to reproduce results.
- **2_BenchmarkData**: Contains the raw performance results of all kernels in CSV and YAML formats.
- **3_LibraryLogic**: Contains the winning (optimal) kernel configurations in YAML format. Typically, rocBLAS takes the YAML files from this folder.
- **4_LibraryClient**: Contains the code objects, kernels, and library code. This is the output of running `TensileCreateLibrary` using the `3_LibraryLogic` directory as an input.

CHAPTER

TWO

INTRODUCTION

Tensile is written in both Python (for library/kernel generation) and C++ (for client headers and library tests)—it is a vital project to the ROCm ecosystem, providing optimized kernels for downstream libraries such as https://github.com/ROCm/rocBLAS.

The parts of Tensile that are written in Python consist of applications that, collectively, are responsible for generating optimized kernels and generating library objects to access these kernels from client code.

CHAPTER

THREE

SOLUTION SELECTION CATALOGS

Tensile provides a mechanism by which only a subset of the code object files produced during a build are loaded at runtime. This is necessary to avoid the overhead associated with loading code object files including initialization time and the memory footprint of the loaded code object files. However, this introduces the problem of knowing which code object file to load. Solution selection is the process by which the **TensileHost** library determines what kernel is preferred and, in turn, what code object file contains the selected kernel. This process uses a hierarchical structure to efficiently search for kernels based on hardware, problem size, and transpose, among others. This is the role of the **solution selection catalog**[1]—a serialized file that uses a hierarchical schema to organize kernel metadata for efficient lookup at runtime.

Note

Throughout this document we will refer to catalog files with the .yaml extension. In practice, solution selection catalogs are usually serialized with MessagePack, which uses the .dat extension.

3.1 Catalog hierarchy

Fig. 3.1: Solution selection catalog heirarchy for gfx900 and gfx90a

Level 1: Hardware

At runtime, only kernels compatible with the device can execute. As such, the top level of the hierarchy involves hardware comparisons using GFX architecture.

Level 2: Operation

This layer is a mapping from a GEMM transpose setting, defined using Einstein tensor notation (e.g. *Contraction_l_Alik_Bjlk_Cijk_Dijk*) to a list of problem properties.

Level 3: Problem

This layer matches against specific problem properties such as input and output types, and features like high precision accumulation and stochastic rounding.

Level 4: Exact solution

Finally, exact solutions contain fine-grained details about each solution that can be used during solution selection to locate the best kernel and to assert that the requested problem predicates are satisfied. Each kernel will have an index and a performance ranking. During solution selection, the highest ranked kernel from this pool will be selected.

[1] Previously these files were called *master solution libraries* because they contain two top level keys, "solutions" and "library". The term *solution selection catalog* was later adopted to clarify the purpose of this file within the larger context of the Tensile C++ API.

3.2 Build modes

Tensile comes equipped with multiple build modes, which affect the way solution selection catalogs are generated.

3.2.1 Mode 1: Lazy library loading

If `--lazy-library-loading` is enabled, then a "parent" catalog is generated for each architecture, named

TensileLibrary_lazy_<gfx>.yaml

This file contains a reference to each of it's "child" catalogs, but doesn't have details about the exact solutions. These settings are instead held in the "child" catalogs, which use the naming convention

TensileLibrary_Type_<precision>_<problem type>_<gfx>.yaml

Here, *precision* is the data type, *problem type* is the GEMM type, including transpose and accumulate settings, and *gfx* is the hardware GFX archiecture.

For example, *TensileLibrary_Type_HH_Contraction_l_Alik_Bjlk_Cijk_Dijk_<gfx>.yaml* identifies a code object library for half precision contractions on two transpose matrices, otherwise known as HGEMM TT. In this way, the child catalogs contain the solution metadata, while the parent catalog is responsible for organizing the child catalogs by hardware, problem type, transpose, precision, and other predicates. This has the benefit of reducing the memory footprint of the calling application, as code object libraries are compiled separately and loaded only when required.

Example: Build outputs

Listing 3.1: Lazy library loading build outputs for *DD_Contraction_l_Alik_Bjlk_Cijk_Dijk*

```
build/
└── library/
    ├── Kernels.so-000-gfx1030.hsaco
    ├── Kernels.so-000-gfx900.hsaco
    ├── Kernels.so-000-gfx906.hsaco
    ├── TensileLibrary_lazy_gfx1030.yaml                    # [A]
    ├── TensileLibrary_lazy_gfx900.yaml
    ├── TensileLibrary_lazy_gfx906.yaml
    ├...
    ├── TensileLibrary_Type_..._fallback_gfx1030.hsaco
    ├── TensileLibrary_Type_..._fallback_gfx900.hsaco
    ├── TensileLibrary_Type_..._fallback_gfx906.hsaco
    ├── TensileLibrary_Type_..._fallback.yaml               # [B]
    ├── TensileLibrary_Type_..._gfx900.co
    ├── TensileLibrary_Type_..._gfx900.hsaco
    ├── TensileLibrary_Type_..._gfx900.yaml                 # [C]
    ├── TensileLibrary_Type_..._gfx906.co
    ├── TensileLibrary_Type_..._gfx906.yaml                 # [D]
```

Line **[A]** shows the parent catalog for gfx1030, the first of the three parent catalogs generated. Line **[B]** shows a fallback child catalog, which reference each of the archiecture specific fallback kernels in the associated .hsaco files. This means that at least some of the parameter/problem type combinations for *DD_Contraction_l_Alik_Bjlk_Cijk_Dijk* haven't been explicitly tuned for these architectures. Note that the matching .hsaco files (above **[B]**) are code object libraries for HIP source kernels. These files are referenced by the fallback catalog. Line **[C]** shows a child catalog for gfx900 that references both HIP source and assembly source kernels, found in the associated .hsaco and .co files, respectively. Line **[D]** shows a child catalog for gfx906, similar to the gfx900 catalog. However, notice that there is only one associated .co file. This means that there are only assembly source kernels in this catalog.

Example: Parent solution selection catalog

Listing 3.2: build/library/TensileLibrary_lazy_gfx900.yaml

```yaml
library:
  rows:                                                    # [A_]
  - library:
      map:
        Contraction_l_Alik_Bjlk_Cijk_Dijk:                 # [B_]
          ...
          rows:                                            # [C_]
          - library: {type: Placeholder, value: TensileLibrary_Type_SS_..._fallback}
            predicate:
              type: And
              value:
              - type: TypesEqual
                value: [Float, Float, Float, Float]
              - {type: HighPrecisionAccumulate, value: false}
              - {type: F32XdlMathOp, value: Float}
              - {type: StochasticRounding, value: false}
          - ...
          type: Problem
          ...
        Contraction_l_Alik_Bljk_Cijk_Dijk:
          rows:
            - ...
          type: Problem                                    # [_C]
      property: {type: OperationIdentifier}
      type: ProblemMap                                     # [_B]
    predicate: {type: TruePred}
  type: Hardware                                           # [_A]
solutions: []
```

Line **[A]** shows the top level of the parent catalog, which contains a single row for each hardware architecture. Line **[B]** shows the problem map for the operation *Contraction_l_Alik_Bjlk_Cijk_Dijk*. Line **[C]** shows the problem type and predicates used to match against exact solutions contained in the child catalogs.

3.2.2 Mode 2: Merge files

> **Warning**
>
> This feature is not recommended and is in the process of being deprecated.

When `--merge-files` is enabled, one solution catalog is generated for each architecture, named

TensileLibrary_<gfx>.yaml

The catalog contains information about supported GEMM types and solution metadata that is used to locate the optimal kernel for a requested GEMM. This pattern has the drawback that all code object libraries are loaded eagerly, thereby increasing both the initialization time and memory footprint of the calling application.

Example

Say you're building libraries for gfx908 and gfx90a with `--merge-files`. The build output directory would look like this

```
build/
└── library/
    ├── Kernels.so-000-gfx1030.hsaco
    ├── Kernels.so-000-gfx1030.hsaco
    ├── Kernels.so-000-gfx1030.hsaco
    ├── Kernels.so-000-gfx900.hsaco
    ├── Kernels.so-000-gfx906.hsaco
    ├── TensileLibrary_gfx1030.co
    ├── TensileLibrary_gfx1030.yaml
    ├── TensileLibrary_gfx900.co
    ├── TensileLibrary_gfx900.yaml
    ├── TensileLibrary_gfx906.co
    └── TensileLibrary_gfx906.yaml
```

CHAPTER

FOUR

TENSILE CLI REFERENCE

The Tensile project provides several command line tools. Here is the standard syntax for the CLI tools:

Table 4.1: Usage syntax for Tensile's command-line documentation

Notation	Description
`<Text inside angle brackets>`	Required argument.
`[Text inside square brackets]`	Optional arguments.
`{Text inside braces}`	Set of arguments. One is required.

Warning

Consider undocumented command-line options experimental or deprecated.

4.1 TensileCreateLibrary

`TensileCreateLibrary` is a command-line tool that generates libraries and code object files for a set of supplied logic files.

4.1.1 Syntax

```
TensileCreateLibrary [OPTIONS...] <LOGIC DIRECTORY> <OUTPUT DIRECTORY> <RUNTIME LANGUAGE>
```

4.1.2 Required arguments

Here is the list of required arguments for invoking the `TensileCreateLibrary` command.

Table 4.2: Required arguments

Argument	Description
<LOGIC DIRECTORY>	Absolute path for logic files. The logic files are generally found in either of the following locations: - The build directory under `3_LibraryLogic`, when generated by the Tensile program. - In a project that hosts pregenerated logic files, such as rocBLAS.
<OUTPUT DIRECTORY>	Absolute or relative path to the output directory where build artifacts are placed.
<RUNTIME LANGUAGE>	Runtime language out of OCL, HIP, or HSA

4.1.3 Options

Here is the list of optional arguments for invoking the `TensileCreateLibrary` command.

Table 4.3: Options

Option	Description
--architecture=A	Architectures to generate a library for. When specifying multiple options, use quoted and semicolon-delimited architectures such as --architecture='gfx908;gfx1012'. Supported architectures include: all; gfx000; gfx803; gfx900; gfx900:xnack-; gfx906; gfx906:xnack+; gfx906:xnack-; gfx908; gfx908:xnack+; gfx908:xnack-; gfx90a; gfx90a:xnack+; gfx90a:xnack-; gfx940; gfx940:xnack+; gfx940:xnack-; gfx941; gfx941:xnack+; gfx941:xnack-; gfx942; gfx942:xnack+; gfx942:xnack-; gfx1010; gfx1011; gfx1012; gfx1030; gfx1031; gfx1032; gfx1034; gfx1035; gfx1100; gfx1101; gfx1102.
--build-client	Builds Tensile client executable that is used for stand alone benchmarking. This option is set by default.
--client-config	Creates `best-solution.ini` in the output directory for the library and generated code object files. This option is set by default.
--code-object-version={defa	HSA code object version.
--cxx-compiler={an hipcc} or {clang++, hipcc} for Windows	C++ compiler used when generating binaries.
--embed-library=EMB	Specifies the library to embed into static variables.
--embed-library-key=EMBED	Access key for embedding library files.
--generate-manifest-and-exit	Similar to the `dry-run` option for `make`, this option computes the outputs of `TensileCreateLibrary` and writes the expected outputs to a manifest file but doesn't execute the commands to generate the output.
--generate-sources-and-exit	Skips building the source and assembly code object files. Outputs source files only and exits.
--ignore-asm-cap-cache	Ignores the asm capability cache and derives the asm capabilities at runtime.
--jobs=CPUTH or -j CPUTHREA	Number of parallel jobs to launch. If this option is supplied with a value higher than `nproc`, the number of parallel jobs will be the same as the number of cores. If this option is supplied with a value below 1 (0 or -1), the number of parallel jobs will be the same as the number of cores, up to a maximum of 64. The default value is -1.
--lazy-library-loading	Loads Tensile libraries only when needed.
--library-format={yam	Specifies the library format to use. Default value: `msgpack`.
--no-enumerate	Prohibits `rocm_agent_enumerator` from running.
--no-merge-files	Stores every solution and kernel in separate files.
--no-short-file-names	Prohibits short files names.
--num-merged-files=NUMM	Number of files the kernels must be written into.
--merge-files	Stores all solutions in a single file. This is set by default.
--short-file-names	Converts solution and kernel names to serial Ids if Windows kernel name is too long. The option is set by default.
--separate-	Separates `TensileLibrary` file according to architecture to reduce the library file loading time

4.1.4 Usage without options

Here is how to run `TensileCreateLibrary` command without options. The specified logic directory contains the logic files and the generated artifacts are directed to the *tensile-output* directory that is created in the directory where the `TensileCreateLibrary` command is invoked. The runtime language is HIP.

```
TensileCreateLibrary /home/myuser/Logic tensile-output HIP
```

4.1.5 Usage with options

Here is how to run `TensileCreateLibrary` command with options. The following example shows options with and without arguments.

```
TensileCreateLibrary --separate-architectures --jobs=32 /home/myuser/Logic tensile-
↪output HIP
```

CHAPTER

FIVE

TENSILE API REFERENCE

This topic provides a categorywise listing of Tensile APIs.

5.1 Common

5.2 EmbeddedData

5.3 TensileCreateLibrary

5.4 Utilities

Tensile.Utilities.Profile.**profile**(*func: Callable*) → Callable

Profiling decorator.

Add `@profile` to mark a function for profiling; set the environment variable TENSILE_PROFILE=ON to enable profiling decorated functions.

Tensile.Utilities.String.**splitDelimitedString**(*input: str*, *delimiters: Set[str]*) → Set[str]

Given the delimiters, split string into a set.

Parameters

- **input** – String to be split.
- **delimiters** – In order of decreasing priority, delimiters to split the string by.

Returns

Input separated by delimiters as a set.

Tensile.Utilities.toFile.**toFile**(*outputFile: Path*, *contents: List[str]*, *delimiter: str* = '\n') → None

Generates a user specified delimited file.

Writes the elements of a List of strings with a given delimiter.

Parameters

- **outputFile** – Path to file for writing manifest.
- **contents** – List of items to write manifest.

- **delimiter** – Symbol used to delimit elements when writing file.

Raises

AssertionError – If contents is not a List[str]

CHAPTER

SIX

ENVIRONMENT VARIABLES

This topic lists the environment variables that enable testing, debugging, and experimental features for Tensile clients and applications.

Table 6.1: Environment variables

Environment variable	Description	Values
TENSILE_DB	Enables debugging features based on the supplied value. TENSILE_DB is a bit field, so options can be set individually or combined. To enable all debug output, set TENSILE_DB=0xFFFF.	0x2 or 0x4 - Prints extra information about the solution selection process. Indicates if a kernel was an exact match, or if a sequence of kernels is considered for a closest match. 0x8 - Prints extra information about the hardware selection process. 0x10 - Prints debug-level information about predicate evaluations. 0x20 - Prints a list of loaded or missing code object libraries. 0x40 - Prints kernel launch arguments, including the kernel name, work group size and count, and all arguments passed. 0x80 - Prints size of allocated tensors. 0x100 - Prints debug information about convolution reference calculations. 0x200 - Prints more detailed information about convolution reference calculations. 0x1000 - Prints information about the loading of embedded, YAML, or MessagePack libraries. 0x4000 - Prints solution lookup efficiency. 0x8000 - Prints the name of selected kernels. 0x80000 - Prints the name of selected kernels and number of common kernel parameters such as Matrix Instruction, MacroTile, ThreadTile, DepthU, and so on.
TENSILE_DB2	Enables extended debugging features based on the supplied value. When enabled, Tensile skips launching kernels for debug purposes, but continues to perform other steps such as kernel selection, data allocation, and initialization.	1 - Enable 2 - Disable
TENSILE_NAIVE_SEARCH	Performs a naive search for matching kernels instead of the standard optimized search.	1 - Enable 2 - Disable
TENSILE_TAM_SELECTION_ENABLE	Enables tile aware solution selection.	1 - Enable 2 - Disable

CHAPTER

SEVEN

NOMENCLATURE

7.1 General Matrix Multiplication

General matrix multiplication (GEMM) is a level 3 BLAS operation that computes the product of two matrices, formalized by the equation,

$$C = \alpha AB + \beta C$$

where α and β are scalars and A and B are optionally transposed input matrices.

Table 7.1: GEMM data types.

Abbreviation	Description	Precision
HGEMM	Half precision general matrix multiplication	16-bit
SGEMM	Single precision general matrix multiplication	32-bit
DGEMM	Double precision general matrix multiplication	64-bit
CGEMM	Single precision complex general matrix multiplication	32-bit
ZGEMM	Double precision complex general matrix multiplication	64-bit

Table 7.2: GEMM operations; N (non-transpose) and T (transpose) represent the transpose state of the input matrices.

Operation	Equation
NN	$C_{i,j} = \sum_l A_{i,l} B_{l,j}$
NT	$C_{i,j} = \sum_l A_{i,l} B_{j,l}$
TN	$C_{i,j} = \sum_l A_{l,i} B_{l,j}$
TT	$C_{i,j} = \sum_l A_{l,i} B_{j,l}$
Batched-GEMM	$C_{i,j,k} = \sum_l A_{i,l,k} B_{l,j,k}$
2D Summation	$C_{i,j} = \sum_{k,l} A_{i,k,l} B_{j,l,k}$
3 Batched Indices	$C_{i,j,k,l,m} = \sum_n A_{i,k,m,l,n} B_{j,k,l,n,m}$
4 Free Indices	$C_{i,j,k,l,m} = \sum_{n,o} A_{i,k,m,o,n} B_{j,m,l,n,o}$

7.2 Indices

The indices describe the dimensionality of the problem to be solved. A GEMM operation takes two 2-dimensional matrices as input, adds up to four input dimensions and contracts them along one dimension. This cancels out two dimensions, leading to a 2-dimensional result. When an index shows up in multiple tensors, those tensors must be the same size along with the dimension, however, they can have different strides.

There are three categories of indices or dimensions used in the problems supported by Tensile: free, batch, and bound. **Tensile only supports problems with at least one pair of free indices.**

7.2.1 Free indices

Free indices are the paired indices of tensor C with one pair in tensor A and another pair in tensor B. i,j,k, and l are the four free indices of tensor C where indices i and k are present in tensor A while indices j and l are present in tensor B.

7.2.2 Batch indices

Batch indices are the indices of tensor C that are present in both tensor A and tensor B. The difference between the GEMM example and the batched-GEMM example is the additional index. In the batched-GEMM example, the index k is the batch index, which batches together multiple independent GEMMs.

7.2.3 Bound indices

The bound indices are also known as summation indices. These indices are not present in tensor C but in the summation symbol (Sum[k]) and in tensors A and B. The inner products (pairwise multiply then sum) are performed along these indices.

CHAPTER

EIGHT

PROGRAMMER'S GUIDE

This topic provides necessary information for programmers interested in contributing to the Tensile source code.

8.1 Development environment

ROCm is the base requirement for contributing to Tensile. See if ROCm is supported on your platform by verifying the supported operating systems list. Then, follow the steps given in the *Installation* guide.

8.1.1 Developing in Docker

ROCm development images are available on Docker Hub for a variety of OS/ROCm versions. See Docker images in the ROCm ecosystem for more details.

8.2 Project structure

Here is the project directory structure to help you find the project files available for contribution.

```
Tensile/
├── Tensile/              Source code, tests, and utilities for the Tensile project
│   └── Tests/                Kernels and application tests
├── HostLibraryTests/     Tests for host-side code running the Tensile library
├── docker/               A collection of useful Dockerfiles
├── docs/                 Documentation source files
├── requirements.txt      Python dependencies for running Tensile applications
├── pytest.ini            Configuration settings for pytest
├── tox.ini               Configuration settings for the Tox environment management tool
└── setup.py              Package build and installation script
```

8.3 Testing

Tensile uses pytest to manage library or kernel tests. The Tensile project utilizes pytest markers to filter the tests to be run. Important markers include `pre_checkin`, `extended`, `integration`, and `unit`. Refer to pytest.ini for all supported markers.

You can run a test via the `tox ci` environment by passing the desired test marker using `-m <MARKER>`:

```
tox run -e ci -- -m {pre_checkin|extended|integration|unit}
```

Note that `--` is used to pass options to the underlying pytest command.

> **Note**
>
> By default, the `tox run` command runs pre-checkin tests, when no markers are specified via `-m`.

8.3.1 Unit tests and coverage reports

All unit tests are available in `Tensile/Tests/unit/`. A convenience command is included to add coverage reporting:

```
tox run -e unittest

# OR for 32 processes

tox run -e unittest -- -n 32
```

By default, coverage results are dumped to the terminal. To generate reports in other formats such as HTML, use:

```
tox run -e unittest -- --cov-report=html
```

Files and directories excluded from coverage reporting are itemized in .coveragerc.

Although, we encourage to run unit tests using `tox` for consistency, you can also run the tests directly using `pytest` for quicker feedback. For example, To run a single test named `test_foo`, use:

```
pytest unit/test_TensileCreateLibrary.py -k "test_foo" --capture=no -v
```

8.3.2 Host library tests

Host library tests ensure that the generated libraries remain operational when called from the client code such as other libraries or applications. These tests are built on gtest. To run them, download the submodule first. Then, from Tensile project's root, run:

```
git submodule update --init
```

Next, you can configure, build, and run the host library tests using any of the following:

- `tox`:

  ```
  tox run -e hostlibtest
  ```

  > **Note**
  >
  > Note that the `tox` command wraps invoke, a tool to manage CLI-invokable tasks. Since tox is fundamentally a Python environment manager and test runner, any reusable shell commands that fall outside its purview are managed by invoke (which are again encapsulated by tox sometimes). See tasks.py for details.

- `invoke`:

  ```
  invoke hostlibtest --configure --build --run
  ```

  Running the preceding command generates an executable `TensileTests`, which can be further used to run the tests.

- Manually: To build and run the tests manually, see the commands in tasks.py. For advanced usage like filtering or repeating test cases, see the gtest documentation.

8.4 Static analysis

8.4.1 Python

To run all static analysis, use the top-level `tox` label `static`:

```
tox run -m static
```

> **Note**
>
> The preceding command might reformat your code, so make sure to commit your changes after running the command.

Linting is evaluated using flake8 and **formatting** is conducted using black and isort. To run a check in isolation, either refer to tox.ini or use one the following commands:

```
tox run -e lint
tox run -e format     # add `-- --check` to check formatting without applying changes
tox run -e isort      # add `-- --check` to check imports without applying changes
```

> **Tip**
>
> To ensure consistent formatting, we recommend you to set up the editor to **format on save** using the same formatter settings as in tox.ini. Either way, ensuring to commit changes after running static analysis reduces wait times caused by simple CI failures.

8.4.2 C++

Formatting is conducted using clang-format. The following command formats all given files, however, we recommend you to setup the editor to *format on save*.

```
clang-format -i style=file <files>
```

Styling rules are configured in .clang-format.

Styling rules are configured in .clang-format.

8.5 Profiling

8.5.1 Python

To enable profiling, use the `@profile` decorator, which must be imported from the `Tensile.Utilities.Profile` module. Under the hood, the decorator wraps the function in a cProfile context and generates a `.prof` file inside the `profiling-results-<date>` directory.

> **Note**
>
> Nested profiling is NOT supported due to the existing limitation with the profiling decorator. This implies that if *func1* calls *func2* in a loop, and both are marked for profiling, the resulting `.prof` file for *func1* will display incorrect results.

8.6 Building documentation

To build the documentation locally, use:

```
tox run -e docs
```

After the documentation is built, the HTML files are generated in `docs/_build/html`.

8.7 Versioning practices

Tensile follows semantic versioning practices such as **major.minor.patch**. See server.org for details.

CHAPTER

NINE

CONTRIBUTION GUIDELINES

This document provides the guidelines for contributing to the Tensile source code.

See also

For information about environment setup and development processes, see *Programmer's guide*.

Tensile's development practice is based on the Gitflow workflow. The **develop** branch is the default branch for development, where all new features and bug fixes are merged. After a PR is merged into **develop**, it undergoes extended testing and profiling. If these checks pass, the PR might be merged into **staging** to be included in the next release. A PR is available in the upcoming release only if it is merged before the release branch is cut.

9.1 Raising issues

To notify us of any existing issue, use the GitHub *Issues* tab.

- Use your best judgment for issue creation. If your issue is already listed, upvote the issue and comment or post to provide additional details, such as how you reproduced this issue.
- If you are not sure of the listed issue being the same as yours, err on the side of caution and file your issue. You can link your issue with the existing issue by providing your issue link and details in the comment section. If your issue is evaluated to be a duplicate, it will be closed.
- If your issue doesn't exist, use the issue template to file a new issue. - When filing an issue, provide as much information as possible including the script output, which is required to collect information about your configuration. This helps to reproduce the issue effectively. - Check your issue regularly, as we might require additional information to successfully reproduce the issue.
- You can also open an issue to ask the maintainers if a proposed change meets the acceptance criteria, or to discuss an idea pertaining to the library.

9.2 Acceptance criteria

Pull Requests (PR) are reviewed by the members of CODEOWNERS.md. Depending on the PR, the reviewers might post comments or request changes. This might require several iterations. The PR is approved only when all the changes requested by the reviewers are marked complete. When a Pull Request is submitted, it undergoes a standard suite of continuous integration tests.

Once the pull request is approved and tests pass, it is merged by a member of the codeowner's community. Attribution for your commit will be preserved when it is merged.

9.3 Submitting a Pull Request

By creating a PR, you agree to the statements made in the *Code License* section. Your PR must target the default *develop* branch, which also serves as our integration branch.

a. **Forking the repository and making changes:**

1. Create a fork of Tensile. Don't create feature branches directly in https://github.com/ROCm/Tensile.
2. Clone your fork locally and set up your *Development environment*.
3. Create your feature branch from **develop** and make changes to the code.
4. Issue `tox run -m precommit` and ensure that all checks pass.
5. Commit your changes using the convention for *Commit messages*.
6. If you are updating documentation, issue `tox run -e docs` and verify the styling and formatting.
7. Push the changes to your fork.

> **Tip**
>
> Keeping the scope of new PRs as narrow as possible improves the chances of it getting accepted. If you are making multiple changes, consider breaking them into separate PRs. Keeping PRs small supports timely code reviews, traceability, and straightforward reversions.

b. **Creating the PR:**

1. Ensure that **your develop** branch is up-to-date with the **upstream develop** branch. This might require a rebase or a merge.
2. Verify that your changes pass static analysis checks and all pre-checkin, host library, and unit tests by running `tox run -m prepr`.
3. Create the PR against the https://github.com/ROCm/Tensile **develop** branch.
4. Fill in as many details as possible. Include description, outcomes, notable changes, and environment information. The availability of information makes the PR review process easier, increasing the likelihood of the PR getting merged in a timely manner.
5. Title the PR in present imperative tense. For example, "*Update* kernel parameters", not "Updates" or "Updated".

> **Tip**
>
> To merge **develop** into your feature branch after a PR is opened, use a merge instead of a rebase.
>
> In general, refrain from force pushing once a feature branch is in PR as it is prone to gotchas in our CI system. Ideally, the git history is linear and clean *before* a PR is created. Hence, we encourage contributors to conduct any rebases or amends prior to opening a PR.

c. **Merging the PR:**

1. Ensure the title of the PR properly describes the changes.
2. Squash and merge the PR. If you are not the maintainer, a maintainer does this for you. When merging multiple changes, use bullet points in the commit message to break down the changes.

9.3.1 Labels

Table 9.1: GitHub PR labels

Label	Effect
ci:profiling	Adds the *profiling* job to the CI pipeline. Profiling artifacts are saved for 10 days.
ci:docs-only	Only runs the *docs/readthedocs* job; omits all other pipeline jobs.

9.4 Coding style and conventions

9.4.1 General conventions

Always use space indentation (four spaces). Never commit a tab (`\t`).

9.4.2 Python doc-strings

Tensile uses autodoc to pull in documentation from doc-strings and integrate them into this site. Use the following guidelines when writing Python functions and modules to maintain quality and consistency.

1. Identify the parameters and returned values with type-hints.
2. For all functions, specify doc-string describing the parameters, return value, and any exception. However, if the function is small and the implementation is straightforward, a one-line doc-string is sufficient.
3. Don't include types directly in the doc-string. Add them as type-hints in the function definition.
4. For doc-string styling, use the Google Python style guide.

9.4.3 Commit messages

1. Use conventional commits.
2. Use the present imperative tense. For example, "add" not "adds" or "added".
3. Don't end the message with a period (.).

9.5 Code license

All code contributed to this project will be licensed under the given LICENSE. Your contribution will be accepted under the same license.

For each new file, include the following licensing header:

```
/*******************************************************************************
 * Copyright (c) 20xx Advanced Micro Devices, Inc.
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in all
 * copies or substantial portions of the Software.
```

(continues on next page)

(continued from previous page)

```
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
 * SOFTWARE.
 *
 *******************************************************************************/
```

9.6 Release cadence

Official Tensile releases are subject to the general ROCm release cadence, which typically follows a quarterly cycle. Latest stable versions of Tensile are available in the **staging** branch.

CHAPTER

TEN

LICENSE

INDEX

P

S

T