
Tensile Documentation

Release 4.43.0

Advanced Micro Devices, Inc.

May 21, 2025

INSTALL

1	Tensile installation	3
1.1	Install ROCm	3
1.2	Install OS dependencies	3
1.3	Install Tensile from source	4
1.4	Running benchmark	4
2	Supported problems	5
2.1	Problem types	5
2.2	Nomenclature	6
3	Tensile CLI reference	7
3.1	TensileCreateLibrary	7
4	Tensile API reference	11
4.1	Common	11
4.2	EmbeddedData	11
4.3	TensileCreateLibrary	12
4.4	Utilities	16
5	Tensile environment variables	19
6	Tensile programmer's guide	21
6.1	Development environment	21
6.2	Project structure	21
6.3	Testing	21
6.4	Static analysis	23
6.5	Profiling	23
6.6	External CI	24
6.7	Building documentation	24
6.8	Versioning practices	24
7	Tensile contribution guidelines	25
7.1	Raising issues	25
7.2	Acceptance criteria	25
7.3	Submitting a Pull Request	26
7.4	Coding style and conventions	27
7.5	Code license	27
7.6	Release cadence	28
8	License	29

Tensile is a tool for creating a benchmark-driven backend library for General Matrix-Matrix Multiplication (GEMM), 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

- *Supported problems*

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.

TENSILE INSTALLATION

You can obtain Tensile as part of ROCm installation or choose to install Tensile from source. This document discusses both the methods to obtain Tensile.

1.1 Install ROCm

Install ROCm for your platform. For installation instructions, refer to [Linux](#) or Windows [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

Here are the steps to install dependencies for Ubuntu distributions. For other distributions, use the appropriate package manager.

Note

These steps might require elevated privileges.

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_sgemv_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:

```
rocm_info | 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_sgemv_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.

SUPPORTED PROBLEMS

This topic discusses the types of problems supported by Tensile.

2.1 Problem types

Here are some of the common problem types supported by Tensile for creating benchmark-driven library.

- Four variants of standard GEMM with two free indices (i,j) and one summation index l.
 - NN where N = Non-transpose:
* $C[i,j] = \text{Sum}[l] A[i,l] * B[l,j]$
 - NT where T = Transpose:
* $C[i,j] = \text{Sum}[l] A[i,l] * B[j, l]$
 - TN:
* $C[i,j] = \text{Sum}[l] A[l, i] * B[l,j]$
 - TT:
* $C[i,j] = \text{Sum}[l] A[l, i] * B[j, l]$
- Batched-GEMM with two free indices, one batched index k, and one summation index l:
 - $C[i,j,k] = \text{Sum}[l] A[i,l,k] * B[l,j,k]$
- 2D summation:
 - $C[i,j] = \text{Sum}[k,l] A[i,k,l] * B[j,l,k]$
- GEMM with three batched indices:
 - $C[i,j,k,l,m] = \text{Sum}[n] A[i,k,m,l,n] * B[j,k,l,n,m]$
- Four free indices, two summation indices, and one batched index:
 - $C[i,j,k,l,m] = \text{Sum}[n,o] A[i,k,m,o,n] * B[j,m,l,n,o]$
- Batched image convolution mapped to 7D tensor contraction:
 - $C[i,j,k,l] = \text{Sum}[m,n] A[i,j,m,n,l] * B[m,n,k,j,l]$

The nomenclature of these problems is explained in the following section.

2.2 Nomenclature

The indices describe the dimensionality of the problem to be solved. A GEMM operation takes two 2-dimensional matrices as input, adding up to four input dimensions and contracts them along one dimension, which 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. These are discussed in the following sections with reference to the examples given in the *problem types* <problem_types>.

2.2.1 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.

2.2.2 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.

2.2.3 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.

2.2.4 Limitation

Tensile supports only problems with at least one pair of free indices.

TENSILE CLI REFERENCE

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

Table 3.1: 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.

3.1 TensileCreateLibrary

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

3.1.1 Syntax

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

3.1.2 Required arguments

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

Table 3.2: Required arguments

Argument	Description
<LOGIC DIREC- TORY>	Absolute path for logic files. The logic files are generally found in either of the following locations: - The build directory under <code>3_LibraryLogic</code> , when generated by the Tensile program. - In a project that hosts pregenerated logic files, such as <code>rocBLAS</code> .
<OUT- PUT DIREC- TORY>	Absolute or relative path to the output directory where build artifacts are placed.
<RUN- TIME LAN- GUAGE>	Runtime language out of OCL, HIP, or HSA

3.1.3 Options

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

Table 3.3: Options

Option	Description
--architecture= =	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 <code>best-solution.ini</code> in the output directory for the library and generated code object files. This option is set by default.
--code-object-version={def:	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 <code>dry-run</code> option for <code>make</code> , this option computes the outputs of <code>TensileCreateLibrary</code> 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 <code>nproc</code> , 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: <code>msgpack</code> .
--no-enumerate	Prohibits <code>rocm_agent_enumerator</code> 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 <code>TensileLibrary</code> file according to architecture to reduce the library file loading time.

3.1.4 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
```

3.1.5 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
```

TENSILE API REFERENCE

This topic provides a categorywise listing of Tensile APIs.

Common

EmbeddedData

TensileCreateLibrary

Utilities

4.1 Common

`Tensile.Common.getArchitectureName(gfxName: str) → str | None`

Maps the provided Gfx architecture to its common name using the **architectureMap**.

Parameters

gfxName – Gfx architecture name to map.

Returns

Common name associated with the provided **gfxName** if it matches or is a substring of a key in **architectureMap**, otherwise None.

`Tensile.Common.tPrint(verbosity: int, arg) → None`

Conditionally prints input to stdout.

If the global print level is greater than or equal to the verbosity, the argument is printed to stdout.

Parameters

- **verbosity** – Level to use for printing arg.
- **arg** – Item to print to stdout.

4.2 EmbeddedData

`Tensile.EmbeddedData.generateLibrary(embedFileName: Path, embedLibraryKey: str, masterFile: Path, cppBaseClass: str, codeObjectFiles: List[str]) → None`

Creates embed library source files.

The goal of an embed library is to convert the master file and code object files into a byte array and inject the byte arrays into a C++ source file. This precludes the need to load data from the master file and code object files at runtime.

Parameters

- **embedFileName** – Name of the generated C++ embed source file without a file extension.

- **embedLibraryKey** – Name or key associated with the embed library.
- **masterFile** – Path to the master library file (.dat or .yaml).
- **cppBaseClass** – Name of type used when specializing the C++ EmbeddedData class template.
- **codeObjectFiles** – List of code object files created by TensileCreateLibrary.

4.3 TensileCreateLibrary

Tensile.TensileCreateLibrary.**addFallback**(*masterLibraries: Dict[str, MasterSolutionLibrary]*) → None
Adds fallback library.

Given a master solution library, add a fallback and if the corresponding architecture is unsupported, replace the library altogether with a fallback.

Parameters

masterLibraries – A dictionary containing the master solution libraries.

Tensile.TensileCreateLibrary.**addNewLibrary**(*masterLibraries: Dict[str, MasterSolutionLibrary]*,
newLibrary: MasterSolutionLibrary, *architectureName: str*)
→ int

Adds new master solution library to a master solution libraries dict.

For a given architecture, add the new library to a dictionary containing libraries for all architectures, compute the starting index for the new library, then remap the indexes for all of the solutions associated with the library.

Parameters

- **masterLibraries** – A dictionary containing all master solution libraries for all architectures.
- **newLibrary** – A master solution library to add to the dictionary.
- **architectureName** – The name of the architecture (or key) associated with the library.

Returns

Index to the last solution of the library associated with current architecture.

Tensile.TensileCreateLibrary.**applyNaming**(*masterLibraries: Dict[str, MasterSolutionLibrary]*) → None
Assigns the solution code object file name for lazy libraries.

Given a master solution library with lazy libraries, assigns the key associated with the lazy library (or name) as the value associated with the corresponding solution's code object file.

Parameters

masterLibraries – A dictionary containing the master solution libraries.

Tensile.TensileCreateLibrary.**filterProcessingErrors**(*kernels: List[Solution]*, *solutions:*
List[Solution], *results: List[Tuple[int, str, str,*
str, str | None]], *errorTolerant: bool*)

Filters out processing errors from lists of kernels, solutions, and results.

This function iterates through the results of **processKernelSource** and identifies any errors encountered during processing. If an error is found (-2 error code), the corresponding kernel, solution, and result are appended to separate lists for removal. After processing, items identified for removal are deleted from the original lists of kernels, solutions, and results.

Parameters

- **kernels** – List of Solution objects representing kernels.

- **solutions** – List of Solution objects associated with kernels.
- **results** – List of tuples representing processing results.
- **printLevel** – Print level indicator.

Returns

Tuple containing filtered lists of kernels, solutions, and results after removing items with processing errors.

Return type

Tuple[List[Solution], List[Solution], List[Any]]

Raises

KeyError – If ‘PrintLevel’ key is not found in the params dictionary.

Tensile.TensileCreateLibrary.**findLogicFiles**(*path*: Path, *logicArchs*: Set[str], *lazyLoading*: bool, *experimentalDir*: str, *extraMatchers*: Set[str] = {'hip'}) → List[str]

Recursively searches the provided path for logic files.

Parameters

- **path** – The path to the directory to search.
- **logicArchs** – Target logic architectures. These are interpreted as filename substrings for which logic files are to be included.
- **extraMatchers** – Additional directories to include for logic files.

Returns

A list of Path objects representing the found YAML files.

Tensile.TensileCreateLibrary.**generateClientConfig**(*outputPath*: Path, *masterFile*: Path, *codeObjectFiles*: List[str], *configFile*: str = 'best-solution.ini') → None

Generates a client config file.

Generates a client config file corresponding to a master library file and code-object parameters created by a TensileCreateLibrary invocation. Also sets best-solution-mode to True.

Parameters

- **outputPath** – The path to the tensile output directory where output files are written.
- **masterFile** – Path to the master library file (.dat or .yaml).
- **codeObjectFiles** – List of code object files created by TensileCreateLibrary.
- **configFile** – Name of config file written to the output directory.

Tensile.TensileCreateLibrary.**generateLogicData**(*logicFiles*: List[str], *version*: str, *printLevel*: int, *separate*: bool) → Dict[str, MasterSolutionLibrary]

Generates a dictionary of master solution libraries.

Parameters

- **logicFiles** – List of paths to logic files.
- **version** – User provided version for the library.
- **printLevel** – Level of debug printing requested.
- **separate** – Separate libraries by architecture.

Returns

For separate architectures, a dictionary of architecture separated master solution libraries; otherwise, a single master solution library for all architectures.

`Tensile.TensileCreateLibrary.generateLazyMasterFileList`(*masterFileList*: `List[Tuple[str, MasterSolutionLibrary]]`) → `List[Tuple[str, MasterSolutionLibrary]]`

Generates a list of tuples that represent the name and the state associated with the lazy master libraries.

This function takes a list of `MasterSolutionLibraries` and traverses each lazy libraries. It collects the items (i.e. the name and corresponding master file) and adds them to list of master files.

Parameters

masterLibraries – A list of name / master solution library pairs.

Returns

List of pairs of master solutions libraries and the corresponding name.

`Tensile.TensileCreateLibrary.generateMasterFileList`(*masterLibraries*: `dict`, *archs*: `List[str]`, *lazy*: `bool`) → `List[Tuple[str, MasterSolutionLibrary]]`

Generates a list of tuples that represent the name and the state associated with the master libraries.

This function takes a dictionary with keys corresponding to a target architecture and values corresponding to the master solution library for that architecture. The function generates a tuple consisting of a `MasterSolutionLibrary` and the associated name. When not separating architectures, the key full will appear in `masterLibraries` indicating that all libraries are combined into a single master library.

Parameters

- **masterLibraries** – A dictionary of architecture name / master solution library pairs.
- **archs** – A list of supported architectures.
- **lazy** – If True, add lazy library master files.

Returns

List of pairs of master solutions libraries and the corresponding name.

`Tensile.TensileCreateLibrary.makeMasterLibraries`(*logicList*: `List[LibraryLogic]`, *separate*: `bool`) → `Dict[str, MasterSolutionLibrary]`

Creates a dictionary of master solution libraries.

Iterates through a list of `LibraryLogic` objects creating master solution libraries and modifying the solution indexing as required.

Parameters

- **logicFiles** – List of `LibraryLogic` objects.
- **separate** – Separate libraries by architecture.

Returns

An architecture separated master solution libraries or a single master solution library for all architectures.

`Tensile.TensileCreateLibrary.makeSolutions`(*masterLibraries*: `dict`, *separate*: `bool`)

Extracts the solutions from the master solution library.

Given a master solution library, forms a flattened generator that yields solutions by iterating over all of the solutions contained in the master solution libraries. If using separate architectures but not using lazy loading, `lazyLibraries` should be an empty dict.

Parameters

masterLibraries – A dictionary containing the master solution libraries.

Returns

Generator representing a sequence of library logic tuples.

`Tensile.TensileCreateLibrary.markDuplicateKernels`(*kernels: List[Solution], kernelWriterAssembly: KernelWriterAssembly*) → `List[Solution]`

Marks duplicate assembly kernels based on their generated base file names.

Kernels written in Assembly language may generate duplicate output file names, leading to potential race conditions. This function identifies such duplicates within the provided list of Solution objects and marks them to prevent issues.

Parameters

kernels – A list of Solution objects representing kernels to be processed.

Returns

A modified list of Solution objects where kernels identified as duplicates are marked with a *duplicate* attribute indicating their duplication status.

Notes

This function sets the “duplicate” attribute on Solution objects, and thereby prepares kernels for **processKernelSource**, which requires “duplicate” to be set.

`Tensile.TensileCreateLibrary.parseLibraryLogicFiles`(*logicFiles: List[str]*) → `List[LibraryLogic]`

Load and parse logic (yaml) files.

Given a list of paths to yaml files containing library logic, load the files into memory and parse the data into a named tuple (i.e. LibraryLogic). This operation is parallelized over N processes.

Parameters

logicFiles – List of paths to logic files.

Returns

List of library logic tuples.

`Tensile.TensileCreateLibrary.prepAsm`(*kernelWriterAssembly: KernelWriterAssembly, isLinux: bool, buildPath: Path, isa: Tuple[int, int, int], printLevel: int*)

Create and prepare the assembly directory; called ONCE per output directory.

This function is called once per output directory. It creates a directory “assembly” under the provided **buildPath**, and generates a bash script for compiling object files into code object files.

Parameters

- **kernelWriterAssembly** – Assembly writer object.
- **buildPath** – Path to directory where assembly files will be written.

`Tensile.TensileCreateLibrary.sanityCheck`(*srcLibPaths: List[str], asmLibPaths: List[str], codeObjectPaths: List[str], genSourcesAndExit: bool*)

Verifies that generated code object paths match associated library paths.

Parameters

- **srcLibPaths** – Source library paths (.hsaco).
- **asmLibPaths** – Assembly library paths (.co).
- **coPaths** – Code object paths containing generated kernels; should contain all assembly and source library paths.

- **genSourcesAndExit** – Flag identifying whether only source file should be generated.

Raises

ValueError – If code object paths do not match library paths.

Tensile.TensileCreateLibrary.**verifyManifest**(*manifest: Path*) → bool

Verifies whether the files listed in the manifest exist on disk.

Parameters

manifest – Path to the manifest file.

Returns

True if all files exist on disk, otherwise False.

Tensile.TensileCreateLibrary.**writeMasterFile**(*libraryPath: Path, format: str, naming: dict, name: str, lib: MasterSolutionLibrary*) → None

Writes a master file to disk as a .yaml or .dat file.

Parameters

- **libraryPath** – Path to library subdirectory located in the tensile output directory.
- **format** – Output format of written file (.dat or .yaml).
- **naming** – Kernel minimum naming.
- **name** – Name of the masterfile.
- **lib** – Master solution library data.

4.4 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 | TextIOWrapper, 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. The state of the file is inspected to determine if it should be opened. If the file is already opened, the file is not closed after writing.

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]

Notes

- If outputFile is a TextIOWrapper, the file must not be closed.
- Providing an open file is useful for writing multiple calls to the same file.

TENSILE ENVIRONMENT VARIABLES

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

Table 5.1: 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.	<p>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.</p> <p>0x8 - Prints extra information about the hardware selection process.</p> <p>0x10 - Prints debug-level information about predicate evaluations.</p> <p>0x20 - Prints a list of loaded or missing code object libraries.</p> <p>0x40 - Prints kernel launch arguments, including the kernel name, work group size and count, and all arguments passed.</p> <p>0x80 - Prints size of allocated tensors.</p> <p>0x100 - Prints debug information about convolution reference calculations.</p> <p>0x200 - Prints more detailed information about convolution reference calculations.</p> <p>0x1000 - Prints information about the loading of embedded, YAML, or MessagePack libraries.</p> <p>0x4000 - Prints solution lookup efficiency.</p> <p>0x8000 - Prints the name of selected kernels.</p> <p>0x80000 - Prints the name of selected kernels and number of common kernel parameters such as Matrix Instruction, MacroTile, ThreadTile, DepthU, and so on.</p>
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.	<p>1 - Enable</p> <p>2 - Disable</p>
TENSILE_NAIVE_SEARCH	Performs a naive search for matching kernels instead of the standard optimized search.	<p>1 - Enable</p> <p>2 - Disable</p>
TENSILE_TAM_SELECTION_ENABLE	Enables tile aware solution selection.	<p>1 - Enable</p> <p>2 - Disable</p>

TENSILE PROGRAMMER'S GUIDE

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

6.1 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 [Tensile installation](#) guide.

6.1.1 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.

6.2 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

6.3 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`.

6.3.1 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
```

6.3.2 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](#).

6.4 Static analysis

6.4.1 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 of 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.

6.4.2 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`.

6.5 Profiling

6.5.1 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.

6.6 External CI

Azure Pipelines are run for every pull request and commit targeting the develop and mainline branches. The pipeline packages up the wheel file and runs `pre_checkin` tests on a `gfx942` system. [Click](#) on the job corresponding to the pull request or commit to view execution logs.

6.7 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`.

6.8 Versioning practices

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

TENSILE 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 *Tensile 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.

7.1 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.

7.2 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.

7.3 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.

7.3.1 Labels

Table 7.1: GitHub PR labels

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

7.4 Coding style and conventions

7.4.1 General conventions

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

7.4.2 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](#).

7.4.3 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 (`.`).

7.5 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.  
*  
*****/
```

7.6 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.

LICENSE

Copyright (C) 2016-2025 Advanced Micro Devices, Inc. All rights reserved.

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.

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.

INDEX

A

addFallback() (in module *Tensile.CreateLibrary*), 12
addNewLibrary() (in module *Tensile.CreateLibrary*), 12
applyNaming() (in module *Tensile.CreateLibrary*), 12

F

filterProcessingErrors() (in module *Tensile.CreateLibrary*), 12
findLogicFiles() (in module *Tensile.CreateLibrary*), 13

G

generateClientConfig() (in module *Tensile.CreateLibrary*), 13
generateLazyMasterFileList() (in module *Tensile.CreateLibrary*), 14
generateLibrary() (in module *Tensile.EmbeddedData*), 11
generateLogicData() (in module *Tensile.CreateLibrary*), 13
generateMasterFileList() (in module *Tensile.CreateLibrary*), 14
getArchitectureName() (in module *Tensile.Common*), 11

M

makeMasterLibraries() (in module *Tensile.CreateLibrary*), 14
makeSolutions() (in module *Tensile.CreateLibrary*), 14
markDuplicateKernels() (in module *Tensile.CreateLibrary*), 15

P

parseLibraryLogicFiles() (in module *Tensile.CreateLibrary*), 15
prepAsm() (in module *Tensile.CreateLibrary*), 15
profile() (in module *Tensile.Utilities.Profile*), 16

S

sanityCheck() (in module *Tensile.CreateLibrary*), 15
splitDelimitedString() (in module *Tensile.Utilities.String*), 16

T

toFile() (in module *Tensile.Utilities.toFile*), 16
tPrint() (in module *Tensile.Common*), 11

V

verifyManifest() (in module *Tensile.CreateLibrary*), 16

W

writeMasterFile() (in module *Tensile.CreateLibrary*), 16