Composable Kernel Documentation

Release 1.1.0

Advanced Micro Devices, Inc.

Aug 11, 2025

INSTALL

The Composable Kernel library provides a programming model for writing performance critical kernels for machine learning workloads across multiple architectures including GPUs and CPUs, through general purpose kernel languages such as HIP C++.

The Composable Kernel repository is located at https://github.com/ROCm/composable_kernel.

Install

- *Composable Kernel prerequisites*
- *Build and install Composable Kernel*
- *Build and install Composable Kernel on a Docker image*

Conceptual

- *Composable Kernel structure*
- *Composable Kernel mathematical basis*

Tutorials

- *Composable Kernel examples and tests*

Reference

- *Composable Kernel supported scalar types*
- *Composable Kernel custom types*
- *Composable Kernel vector utilities*
- *Composable Kernel wrapper*
- Composable Kernel API reference
- CK Tile API reference
- *Composable Kernel complete API class list*

To contribute to the documentation refer to Contributing to ROCm.

You can find licensing information on the Licensing page.

CHAPTER

ONE

COMPOSABLE KERNEL PREREQUISITES

Docker images that include all the required prerequisites for building Composable Kernel are available on Docker Hub.

The following prerequisites are required to build and install Composable Kernel:

- cmake
- hip-rocclr
- iputils-ping
- jq
- libelf-dev
- libncurses5-dev
- libnuma-dev
- libpthread-stubs0-dev
- llvm-amdgpu
- mpich
- net-tools
- python3
- python3-dev
- python3-pip
- redis
- rocm-llvm-dev
- zlib1g-dev
- libzstd-dev
- openssh-server
- clang-format-12

CHAPTER

TWO

BUILDING AND INSTALLING COMPOSABLE KERNEL WITH CMAKE

Before you begin, clone the Composable Kernel GitHub repository and create a `build` directory in its root:

```
git clone https://github.com/ROCm/composable_kernel.git
cd composable_kernel
mkdir build
```

Change directory to the `build` directory and generate the makefile using the `cmake` command. Two build options are required:

- `CMAKE_PREFIX_PATH`: The ROCm installation path. ROCm is installed in `/opt/rocm` by default.
- `CMAKE_CXX_COMPILER`: The path to the Clang compiler. Clang is found at `/opt/rocm/llvm/bin/clang++` by default.

```
cd build
cmake ../. -D CMAKE_PREFIX_PATH="/opt/rocm" -D CMAKE_CXX_COMPILER="/opt/rocm/llvm/bin/
↪clang++" [-D<OPTION1=VALUE1> [-D<OPTION2=VALUE2>] ...]
```

Other build options are:

- `DISABLE_DL_KERNELS`: Set this to "ON" to not build deep learning (DL) and data parallel primitive (DPP) instances.

> **Note**
>
> DL and DPP instances are useful on architectures that don't support XDL or WMMA.

- `CK_USE_FP8_ON_UNSUPPORTED_ARCH`: Set to `ON` to build FP8 data type instances on gfx90a without native FP8 support.
- `GPU_TARGETS`: Target architectures. Target architectures in this list must all be different versions of the same architectures. Enclose the list of targets in quotation marks. Separate multiple targets with semicolons (`;`). For example, `cmake -D GPU_TARGETS="gfx908;gfx90a"`. This option is required to build tests and examples.
- `GPU_ARCHS`: Target architectures. Target architectures in this list are not limited to different versions of the same architectures. Enclose the list of targets in quotation marks. Separate multiple targets with semicolons (`;`). For example, `cmake -D GPU_TARGETS="gfx908;gfx1100"`.
- `CMAKE_BUILD_TYPE`: The build type. Can be `None`, `Release`, `Debug`, `RelWithDebInfo`, or `MinSizeRel`. CMake will use `Release` by default.

> **Note**
>
> If neither `GPU_TARGETS` nor `GPU_ARCHS` is specified, Composable Kernel will be built for all targets supported by the compiler.

Build Composable Kernel using the generated makefile. This will build the library, the examples, and the tests, and save them to `bin`.

```
make -j20
```

The `-j` option speeds up the build by using multiple threads in parallel. For example, `-j20` uses twenty threads in parallel. On average, each thread will use 2GB of memory. Make sure that the number of threads you use doesn't exceed the available memory in your system.

Using `-j` alone will launch an unlimited number of threads and is not recommended.

Install the Composable Kernel library:

```
make install
```

After running `make install`, the Composable Kernel files will be saved to the following locations:

- Library files: `/opt/rocm/lib/`
- Header files: `/opt/rocm/include/ck/` and `/opt/rocm/include/ck_tile/`
- Examples, tests, and ckProfiler: `/opt/rocm/bin/`

For information about ckProfiler, see the ckProfiler readme file.

For information about running the examples and tests, see *Composable Kernel examples and tests*.

CHAPTER

THREE

COMPOSABLE KERNEL DOCKER CONTAINERS

Docker images that include all the required prerequisites for building Composable Kernel are available on Docker Hub.

The images also contain ROCm, CMake, and the ROCm LLVM compiler infrastructure.

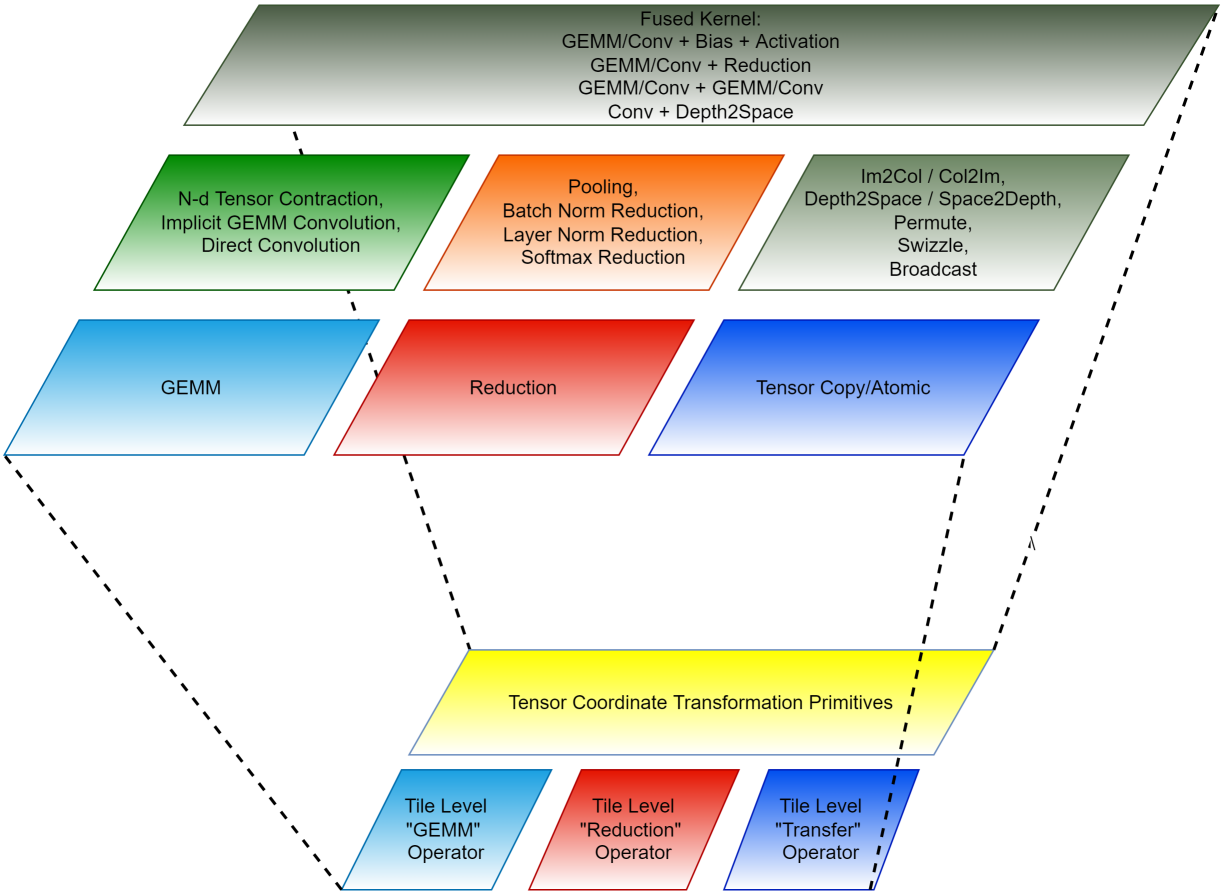
Composable Kernel Docker images are named according to their operating system and ROCm version. For example, a Docker image named `ck_ub22.04_rocm6.3` would correspond to an Ubuntu 22.04 image with ROCm 6.3.

CHAPTER

FOUR

COMPOSABLE KERNEL STRUCTURE

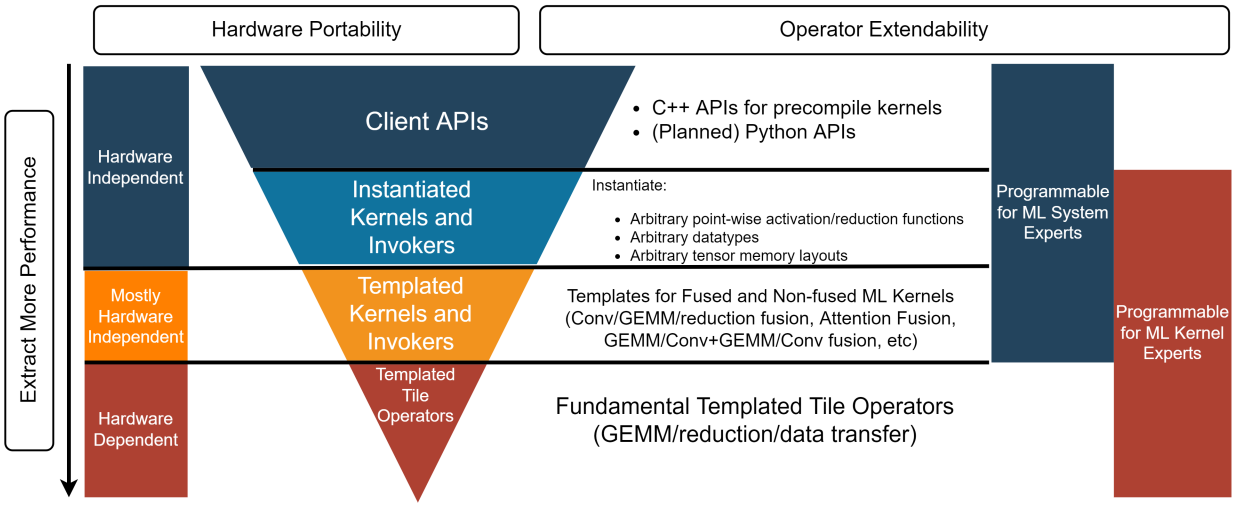
The Composable Kernel library uses a tile-based programming model and tensor coordinate transformation to achieve performance portability and code maintainability. Tensor coordinate transformation is a complexity reduction technique for complex machine learning operators.



The Composable Kernel library consists of four layers:

- a templated tile operator layer
- a templated kernel and invoker layer
- an instantiated kernel and invoker layer
- a client API layer.

A wrapper component is included to simplify tensor transform operations.



CHAPTER

FIVE

COMPOSABLE KERNEL MATHEMATICAL BASIS

This is an introduction to the math which underpins the algorithms implemented in Composable Kernel.

For vectors $x^{(1)}, x^{(2)}, \ldots, x^{(T)}$ of size B you can decompose the softmax of concatenated $x = [x^{(1)} \mid \ldots \mid x^{(T)}]$ as,

$$m(x) = m([x^{(1)} \mid \ldots \mid x^{(T)}]) = \max(m(x^{(1)}), \ldots, m(x^{(T)})) \tag{5.1}$$

$$f(x) = [\exp(m(x^{(1)}) - m(x))f(x^{(1)}) \mid \ldots \mid \exp(m(x^{(T)}) - m(x))f(x^{(T)})] \tag{5.2}$$

$$z(x) = \exp(m(x^{(1)}) - m(x))\ z(x^{(1)}) + \ldots + \exp(m(x^{(T)}) - m(x))\ z(x^{(1)}) \tag{5.3}$$

$$\text{softmax}(x) = f(x)\ /\ z(x) \tag{5.4}$$

where $f(x^{(j)}) = \exp(x^{(j)} - m(x^{(j)}))$ is of size B and $z(x^{(j)}) = f(x_1^{(j)}) + \ldots + f(x_B^{(j)})$ is a scalar.

For a matrix X composed of $T_r \times T_c$ tiles, X_{ij}, of size $B_r \times B_c$ you can compute the row-wise softmax as follows.

For j from 1 to T_c, and i from 1 to T_r calculate,

$$\tilde{m}_{ij} = \text{rowmax}(X_{ij}) \tag{5.5}$$

$$\tilde{P}_{ij} = \exp(X_{ij} - \tilde{m}_{ij}) \tag{5.6}$$

$$\tilde{z}_{ij} = \text{rowsum}(P_{ij}) \tag{5.7}$$

$$\tag{5.8}$$

If $j = 1$, initialize running max, running sum, and the first column block of the output,

$$m_i = \tilde{m}_{i1} \tag{5.9}$$

$$z_i = \tilde{z}_{i1} \tag{5.10}$$

$$\tilde{Y}_{i1} = (\tilde{z}_{ij})^{-1}\tilde{P}_{i1} \tag{5.11}$$

Else if $j > 1$,

1. Update running max, running sum and column blocks $k = 1$ to $k = j - 1$

$$m_i^{new} = \max(m_i, \tilde{m}_{ij}) \tag{5.12}$$

$$z_i^{new} = \exp(m_i - m_i^{new})\ z_i + \exp(\tilde{m}_{ij} - m_i^{new})\ \tilde{z}_{ij} \tag{5.13}$$

$$Y_{ik} = (z_i^{new})^{-1}(z_i)\exp(m_i - m_i^{new})\ Y_{ik} \tag{5.14}$$

2. Initialize column block j of output and reset running max and running sum variables:

$$\tilde{Y}_{ij} = (z_i^{new})^{-1}\exp(\tilde{m}_{ij} - m_i^{new})\tilde{P}_{ij} \tag{5.15}$$

$$z_i = z_i^{new} \tag{5.16}$$

$$m_i = m_i^{new} \tag{5.17}$$

$$\tag{5.18}$$

CHAPTER

SIX

COMPOSABLE KERNEL EXAMPLES AND TESTS

After *building and installing Composable Kernel*, the examples and tests will be moved to `/opt/rocm/bin/`.

All tests have the prefix `test` and all examples have the prefix `example`.

Use `ctest` with no arguments to run all examples and tests, or use `ctest -R` to run a single test. For example:

```
ctest -R test_gemm_fp16
```

Examples can be run individually as well. For example:

```
./bin/example_gemm_xdl_fp16 1 1 1
```

For instructions on how to run individual examples and tests, see their README files in the `example` and `test` GitHub folders.

To run smoke tests, use `make smoke`.

To run regression tests, use `make regression`.

In general, tests that run for under thirty seconds are included in the smoke tests and tests that run for over thirty seconds are included in the regression tests.

CHAPTER

SEVEN

COMPOSABLE KERNEL SUPPORTED SCALAR DATA TYPES

The Composable Kernel library provides support for the following scalar data types:

Type	Bit Width	Description
`double`	64-bit	Standard IEEE 754 double precision floating point
`float`	32-bit	Standard IEEE 754 single precision floating point
`int32_t`	32-bit	Standard signed 32-bit integer
`int8_t`	8-bit	Standard signed 8-bit integer
`uint8_t`	8-bit	Standard unsigned 8-bit integer
`bool`	1-bit	Boolean type
`ck::half_t`	16-bit	IEEE 754 half precision floating point with 5 exponent bits, 10 mantissa bits, and 1 sign bit
`ck::bhalf_t`	16-bit	Brain floating point with 8 exponent bits, 7 mantissa bits, and 1 sign bit
`ck::f8_t`	8-bit	8-bit floating point (E4M3 format) with 4 exponent bits, 3 mantissa bits, and 1 sign bit
`ck::bf8_t`	8-bit	8-bit brain floating point (E5M2 format) with 5 exponent bits, 2 mantissa bits, and 1 sign bit
`ck::f4_t`	4-bit	4-bit floating point format (E2M1 format) with 2 exponent bits, 1 mantissa bit, and 1 sign bit
`ck::f6_t`	6-bit	6-bit floating point format (E2M3 format) with 2 exponent bits, 3 mantissa bits, and 1 sign bit
`ck::bf6_t`	6-bit	6-bit brain floating point format (E3M2 format) with 3 exponent bits, 2 mantissa bits, and 1 sign bit

CHAPTER

EIGHT

COMPOSABLE KERNEL CUSTOM DATA TYPES

Composable Kernel supports the use of custom types that provide a way to implement specialized numerical formats.

To use custom types, a C++ type that implements the necessary operations for tensor computations needs to be created. These should include:

- Constructors and initialization methods
- Arithmetic operators if the type will be used in computational operations
- Any conversion functions needed to interface with other parts of an application

For example, to create a complex half-precision type:

```
struct complex_half_t
{
    half_t real;
    half_t img;
};

struct complex_half_t
{
    using type = half_t;
    type real;
    type img;

    complex_half_t() : real{type{}}, img{type{}} {}
    complex_half_t(type real_init, type img_init) : real{real_init}, img{img_init} {}
};
```

Custom types can be particularly useful for specialized applications such as complex number arithmetic, custom quantization schemes, or domain-specific number representations.

CHAPTER

NINE

COMPOSABLE KERNEL VECTOR TEMPLATE UTILITIES

Composable Kernel includes template utilities for creating vector types with customizable widths. These template utilities also flatten nested vector types into a single, wider vector, preventing the creation of vectors of vectors.

Vectors composed of supported scalar and custom types can be created with the `ck::vector_type` template.

For example, `ck::vector_type<float, 4>` creates a vector composed of four floats and `ck::vector_type<ck::half_t, 8>` creates a vector composed of eight half-precision scalars.

For vector operations to be valid, the underlying types must be either a *supported scalar type* or *a custom type* that implements the required operations.

CHAPTER

TEN

COMPOSABLE KERNEL WRAPPER

The Composable Kernel library provides a lightweight wrapper to simplify the more complex operations.

Example:

```
const auto shape_4x2x4         = ck::make_tuple(4, ck::make_tuple(2, 4));
const auto strides_s2x1x8      = ck::make_tuple(2, ck::make_tuple(1, 8));
const auto layout = ck::wrapper::make_layout(shape_4x2x4, strides_s2x1x8);

std::array<ck::index_t, 32> data;
auto tensor = ck::wrapper::make_tensor<ck::wrapper::MemoryTypeEnum::Generic>(&data[0],␣
↪layout);

for(ck::index_t w = 0; w < size(tensor); w++) {
    tensor(w) = w;
}

// slice() == slice(0, -1) (whole dimension)
auto tensor_slice = tensor(ck::wrapper::slice(1, 3), ck::make_tuple(ck::wrapper::slice(),
↪ ck::wrapper::slice()));
std::cout << "dims:2,(2,4) strides:2,(1,8)" << std::endl;
for(ck::index_t h = 0; h < ck::wrapper::size<0>(tensor_slice); h++)
{
    for(ck::index_t w = 0; w < ck::wrapper::size<1>(tensor_slice); w++)
    {
        std::cout << tensor_slice(h, w) << " ";
    }
    std::cout << std::endl;
}
```

Output:

```
dims:2,(2,4) strides:2,(1,8)
1 5 9 13 17 21 25 29
2 6 10 14 18 22 26 30
```

Tutorials:

- GEMM tutorial

Advanced examples:

- Image to column
- Basic gemm

- Optimized gemm

10.1 Layout

template<typename **Shape**, typename **UnrolledDescriptorType**>

struct **Layout**

Layout wrapper that performs the tensor descriptor logic.

Template Parameters

- **Shape** – Tuple of Number<> (for compile-time layout) or index_t (dynamic layout). It is possible to pass nested shapes (e.g. ((4, 2), 2)), nested dimensions are merged.
- **UnrolledDescriptorType** – *Tensor* descriptor for unnested shape dims.

10.2 Layout helpers

Functions

template<typename **Shape**, typename **Strides**>
__host__ __device__ constexpr auto **make_layout**(const *Shape* &shape, const *Strides* &strides)

Make layout function.

Template Parameters

- **Shape** – Shape for layout.
- **Strides** – Strides for layout.

Returns
Constructed layout.

template<typename **Shape**>
__host__ __device__ constexpr auto **make_layout**(const *Shape* &shape)

Make layout function with packed strides (column-major).

Template Parameters
Shape – Shape for layout.

Returns
Constructed layout.

template<typename **T**>
__host__ __device__ constexpr *T* **get**(const *T* &dim)

Get dim.

Parameters
dim – Dimension.

Returns
Returned the same dimension.

template<index_t **idx**, typename ...**Dims**>
__host__ __device__ constexpr auto **get**(const Tuple<*Dims*...> &tuple)

Get element from tuple (Shape/Strides/Idxs).

Template Parameters
idx – Index to lookup.

Parameters
tuple – Tuple to lookup.

Returns
Requsted element.

template<index_t **idx**, typename **Shape**, typename **UnrolledDesc**>
__host__ __device__ constexpr auto **get**(const *Layout*<*Shape*, *UnrolledDesc*> &layout)

Get sub layout.

Template Parameters
idx – Index to lookup.

Parameters
layout – *Layout* to create sub layout.

Returns
Requsted sub layout.

template<index_t **Idx**, index_t... **Idxs**, typename **T**>
__host__ __device__ constexpr auto **get**(const *T* &elem)

Hierarchical get.

Template Parameters
Idxs – Indexes to lookup.

Parameters
elem – Element to lookup.

Returns
Requsted element.

template<typename **T**>
__host__ __device__ constexpr *T* **size**(const *T* &dim)

Get size.

Parameters
dim – Size.

Returns
Returned the same size.

template<index_t **idx**, typename **Shape**, typename **UnrolledDescriptorType**>
__host__ __device__ constexpr auto **size**(const *Layout*<*Shape*, *UnrolledDescriptorType*> &layout)

Length get (product if tuple).

Template Parameters
idx – Index to lookup.

Parameters
layout – *Layout* to get Shape of.

Returns
Requsted length.

template<typename ...**ShapeDims**>
__host__ __device__ constexpr auto **size**(const Tuple<*ShapeDims*...> &shape)

Shape size (product of dims).

Parameters
shape – Shape to lookup.

Returns
Requsted size.

template<typename **Shape**, typename **UnrolledDescriptorType**>
__host__ __device__ constexpr auto **size**(const *Layout*<*Shape*, *UnrolledDescriptorType*> &layout)

Layout size (product of dims).

Parameters
layout – *Layout* to calculate shape size.

Returns
Requsted size.

template<index_t **idx**, typename ...**Ts**>
__host__ __device__ constexpr auto **size**(const Tuple<*Ts*...> &tuple)

Length get from tuple (product if tuple).

Template Parameters
idx – Index to lookup.

Parameters
tuple – Tuple to lookup.

Returns
Requsted length.

template<index_t **Idx**, index_t... **Idxs**, typename **T**>
__host__ __device__ constexpr auto **size**(const *T* &elem)

Hierarchical size.

Template Parameters

- **Idx** – First index to lookup (to avoid empty Idxs).
- **Idxs** – Next indexes to lookup.

Parameters
elem – Element to lookup.

Returns
Requsted element.

template<typename **Shape**, typename **UnrolledDescriptorType**>
__host__ __device__ constexpr auto **rank**([[maybe_unused]] const *Layout*<*Shape*, *UnrolledDescriptorType*> &layout)

Get layout rank (num elements in shape).

Parameters
layout – *Layout* to calculate rank.

Returns
Requsted rank.

template<typename ...**Dims**>
__host__ __device__ constexpr auto **rank**([[maybe_unused]] const Tuple<*Dims*...> &tuple)

Get tuple rank (num elements in tuple). Return 1 if scalar passed.

Parameters
tuple – Tuple to calculate rank.

Returns
Requsted rank.

template<index_t **IDim**>
__host__ __device__ constexpr index_t **rank**([[maybe_unused]] const Number<*IDim*> &dim)

Rank for scalar.

Parameters
dim – Dimension scalar.

Returns
Returned 1.

__host__ __device__ constexpr index_t **rank**([[maybe_unused]] const index_t &dim)

Rank for scalar.

Parameters
dim – Dimension scalar.

Returns
Returned 1.

template<index_t... **Idxs**, typename **T**>
__host__ __device__ constexpr auto **rank**(const *T* &elem)

Hierarchical rank.

Template Parameters
Idxs – Indexes to lookup.

Parameters
elem – Element to lookup.

Returns
Requsted rank.

template<typename **Shape**, typename **UnrolledDescriptorType**>
__host__ __device__ constexpr auto **depth**(const *Layout*<*Shape*, *UnrolledDescriptorType*> &layout)

Get depth of the layout shape (return 0 if scalar).

Parameters
layout – *Layout* to calculate depth.

Returns
Requsted depth.

template<typename ...**Dims**>
__host__ __device__ constexpr auto **depth**(const Tuple<*Dims*...> &tuple)

Get depth of the tuple. (return 0 if scalar)

Parameters
tuple – Tuple to calculate depth.

Returns
Requsted depth.

template<index_t **IDim**>
__host__ __device__ constexpr index_t **depth**([[maybe_unused]] const Number<*IDim*> &dim)

Depth for scalar.

Parameters
dim – Scalar.

Returns
Returned 0.

__host__ __device__ constexpr index_t **depth**([[maybe_unused]] const index_t &dim)

Depth for scalar.

Parameters
dim – Scalar.

Returns
Returned 0.

template<index_t... **Idxs**, typename **T**>
__host__ __device__ constexpr auto **depth**(const *T* &elem)

Hierarchical depth.

Template Parameters
Idxs – Indexes to lookup.

Parameters
elem – Element to lookup.

Returns
Requsted depth.

template<typename **LayoutType**>
__host__ __device__ constexpr const auto &**shape**(const *LayoutType* &layout)

Get *Layout* shape.

Parameters
layout – *Layout* to get shape from.

Returns
Requsted shape.

template<typename **Shape**, typename **UnrolledDesc**, typename **TileLengths**>
__host__ __device__ constexpr auto **pad**(const *Layout*<*Shape*, *UnrolledDesc*> &layout, const *TileLengths* &tile_lengths)

Pad layout shapes to be adjusted to tile lengths.

Parameters

- **layout** – *Layout* to pad.
- **tile_lengths** – Tile lengths to align layout shape.

Returns
Padded layout.

template<index_t **Idx**, typename **Shape**, typename **UnrolledDesc**, typename **NewLengths**, typename **NewIdxs**>
__host__ __device__ constexpr auto **unmerge**(const *Layout*<*Shape*, *UnrolledDesc*> &layout, const *NewLengths* &new_lengths, [[maybe_unused]] const *NewIdxs* &new_indexes)

Unmerge selected dim in layout.

Template Parameters
Idx – Index to dimension being unmerged.

Parameters

- **layout** – *Layout* to pad.
- **new_lengths** – Dimensions into which the indicated dimension will be divided.
- **new_indexes** – Indexes to shuffle dims. Dims for unmerged dim should be nested.

Returns
Unmerged layout.

10.3 Tensor

template<typename **T**>

struct **Tensor**

Tensor wrapper that performs static and dynamic buffer logic. The tensor is based on a descriptor stored in the *Layout*. Additionally, tensor can be sliced or shifted using multi-index offset.

Template Parameters

- **BufferAddressSpace** – Memory type (Generic, Global, LDS, VGPR, SGPR).
- **ElementType** – Element data type.
- **Shape** – *Tensor* shape (layout component).
- **UnrolledDescriptorType** – Flatten descriptor (layout component).

10.4 Tensor helpers

Typedefs

using **MemoryTypeEnum** = AddressSpaceEnum

Memory type, allowed members:

- Generic,
- Global,
- Lds,
- Sgpr,
- Vgpr,

Functions

template<*MemoryTypeEnum* **MemoryType**, typename **ElementType**, typename **Shape**, typename **UnrolledDescriptorType**>
constexpr auto **make_tensor**(*ElementType* *pointer, const *Layout*<*Shape*, *UnrolledDescriptorType*> &layout)

Make tensor function.

Template Parameters
MemoryType – Type of memory.

Parameters

- **pointer** – Pointer to the memory.
- **layout** – *Tensor* layout.

Returns
Constructed tensor.

template<*MemoryTypeEnum* **MemoryType**, typename **ElementType**, typename **Shape**, typename **UnrolledDescriptorType**>
constexpr auto **make_register_tensor**(const *Layout*<*Shape*, *UnrolledDescriptorType*> &layout)

Make SGPR or VGPR tensor function.

Template Parameters

- **MemoryType** – Type of memory.
- **ElementType** – Memory data type.

Returns
Constructed tensor.

template<*MemoryTypeEnum* **BufferAddressSpace**, typename **ElementType**, typename **Shape**, typename **UnrolledDescriptorType**>
__host__ __device__ void **clear**(*Tensor*<*BufferAddressSpace*, *ElementType*, *Shape*, *UnrolledDescriptorType*> &tensor)

Clear tensor. (Only for Vpgr/Sgpr)

Parameters
tensor – *Tensor* to be cleared.

template<*MemoryTypeEnum* **BufferAddressSpace**, typename **ElementType**, typename **Shape**, typename **UnrolledDescriptorType**>
__host__ __device__ constexpr const auto &**layout**(const *Tensor*<*BufferAddressSpace*, *ElementType*, *Shape*, *UnrolledDescriptorType*> &tensor)

Get *Tensor Layout*.

Parameters
tensor – *Tensor* to get layout of.

Returns
Requsted layout.

template<index_t... **Idxs**, *MemoryTypeEnum* **BufferAddressSpace**, typename **ElementType**, typename **Shape**, typename **UnrolledDescriptorType**>
__host__ __device__ constexpr auto **size**(const *Tensor*<*BufferAddressSpace*, *ElementType*, *Shape*, *UnrolledDescriptorType*> &tensor)

Product of tensor shape dims.

Template Parameters
Idxs – Indexes to access specific shape dim (optional).

Parameters
tensor – *Tensor* to get Shape of.

Returns
Requsted size.

template<index_t... **Idxs**, *MemoryTypeEnum* **BufferAddressSpace**, typename **ElementType**, typename **Shape**, typename **UnrolledDescriptorType**>
__host__ __device__ constexpr auto **rank**(const *Tensor*<*BufferAddressSpace*, *ElementType*, *Shape*, *UnrolledDescriptorType*> &tensor)

Rank of Shape tuple.

Template Parameters
Idxs – Indexes to access specific shape dim (optional).

Parameters
tensor – *Tensor* to get rank of.

Returns
Requsted rank.

template<index_t... **Idxs**, *MemoryTypeEnum* **BufferAddressSpace**, typename **ElementType**, typename **Shape**, typename **UnrolledDescriptorType**>
__host__ __device__ constexpr auto **depth**(const *Tensor*<*BufferAddressSpace*, *ElementType*, *Shape*, *UnrolledDescriptorType*> &tensor)

Depth of Shape tuple.

Template Parameters
Idxs – Indexes to access specific shape dim (optional).

Parameters
tensor – *Tensor* to get depth of.

Returns
Requsted depth.

template<*MemoryTypeEnum* **BufferAddressSpace**, typename **ElementType**, typename **Shape**, typename **UnrolledDescriptorType**>
__host__ __device__ constexpr const auto &**shape**(const *Tensor*<*BufferAddressSpace*, *ElementType*, *Shape*, *UnrolledDescriptorType*> &tensor)

Get *Tensor* shape.

Parameters
tensor – *Tensor* to get shape from.

Returns
Requsted shape.

template<typename **FromType**, typename **ToType**>
constexpr auto **slice**(const *FromType* from, const *ToType* to)

Get dim slice.

Parameters

- **from** – Beginning of the interval.
- **to** – End of the interval. (could be also negative to index from the end)

Returns
Requested slice. Could be used to create sliced tensor from other tensor.

template<typename **ToType**>
constexpr auto **slice**(const *ToType* to)

Get dim slice. (Assumed that from is equal to 1)

Parameters
to – End of the interval. (could be also negative to index from the end)

Returns
Requested slice. Could be used to create sliced tensor from other tensor.

constexpr auto **slice**()

Get whole dim slice (from = 0, to = -1).

Returns
Requested slice. Could be used to create sliced tensor from other tensor.

Functions

template<typename **TensorType**, typename **ThreadShape**, typename **ThreadUnrolledDesc**, typename **ProjectionTuple**>
__host__ __device__ constexpr auto **make_local_partition**(*TensorType* &tensor, [[maybe_unused]] const *Layout*<*ThreadShape*, *ThreadUnrolledDesc*> &thread_layout, const index_t thread_id, const *ProjectionTuple* &projection)

Create local partition for thread (At now only packed partition is supported).

Parameters

- **tensor** – *Tensor* for partition.
- **thread_layout** – *Layout* of threads (could not be transformed).
- **thread_id** – Thread index represented as integer.
- **projection** – Projection is used to remove selected dim from partitioning. Use `slice(X)` to remove dimension, where X is dim size. Use `Number<1>{}` to keep it.

Returns
Partition tensor.

template<typename **TensorType**, typename **ThreadShape**, typename **ThreadUnrolledDesc**>
__host__ __device__ constexpr auto **make_local_partition**(*TensorType* &tensor, const *Layout*<*ThreadShape*, *ThreadUnrolledDesc*> &thread_lengths, const index_t thread_id)

Create local partition for thread (At now only packed partition is supported).

Parameters

- **tensor** – *Tensor* for partition.
- **thread_lengths** – *Layout* of threads (could not be nested).
- **thread_id** – Thread index represented as integer.

Returns
Partition tensor.

template<typename **TensorType**, typename **BlockShapeTuple**, typename **BlockIdxs**, typename **ProjectionTuple**>
__host__ __device__ constexpr auto **make_local_tile**(const *TensorType* &tensor, const *BlockShapeTuple* &tile_shape, const *BlockIdxs* &block_idxs, const *ProjectionTuple* &projection)

Create local tile for thread block. (At now only packed tile is supported).

> **Note**
>
> Temporary to gain the best performance use 2d tile_shape.

Parameters

- **tensor** – *Tensor* for partition.
- **tile_shape** – Shapes of requested tile.
- **block_idxs** – Tuple of block indexes represented as integer. If slice, then get whole dim.

- **projection** – Projection is used to remove selected dim from partitioning. Use `slice(X)` to remove dimension, where X is dim size. Use `Number<1>{}` to keep it.

Returns
Tile tensor.

template<typename **TensorType**, typename **BlockShapeTuple**, typename **BlockIdxs**>
__host__ __device__ constexpr auto **make_local_tile**(const *TensorType* &tensor, const *BlockShapeTuple* &tile_shape, const *BlockIdxs* &block_idxs)

Create local tile for thread block. (At now only packed tile is supported).

> **Note**
> Currently to get the best performance please use 2d shape.

Parameters

- **tensor** – *Tensor* for partition.
- **tile_shape** – Shapes of requested tile.
- **block_idxs** – Tuple of block indexes represented as integer. If slice, then get whole dim.

Returns
Tile tensor.

10.5 Operations

Functions

template<typename **DimAccessOrderTuple**, index_t **VectorDim**, index_t **ScalarPerVector**, typename **SrcTensorType**, typename **DstTensorType**>
__device__ void **copy**(const *SrcTensorType* &src_tensor, *DstTensorType* &dst_tensor)

Perform optimized copy between two tensors partitions (threadwise copy). Tensors must have the same size.

Template Parameters

- **DimAccessOrderTuple** – Tuple with dimension access order.
- **VectorDim** – Dimension for vectorized read and write.
- **ScalarPerVector** – Number of scalar per vectorized read and write.

Parameters

- **src_tensor** – Source tensor.
- **dst_tensor** – Destination tensor.

template<typename **SrcTensorType**, typename **DstTensorType**>
__host__ __device__ void **copy**(const *SrcTensorType* &src_tensor, *DstTensorType* &dst_tensor)

Perform generic copy between two tensors partitions (threadwise copy). Tensors must have the same size.

Parameters

- **src_tensor** – Source tensor.
- **dst_tensor** – Destination tensor.

template<typename **DimAccessOrderTuple**, index_t **VectorDim**, index_t **ScalarPerVector**, typename **SrcTensorType**, typename **DstTensorType**, typename **ThreadShape**, typename **ThreadUnrolledDesc**>
__device__ void **blockwise_copy**(const *SrcTensorType* &src_tensor, *DstTensorType* &dst_tensor, [[maybe_unused]] const *Layout*<*ThreadShape*, *ThreadUnrolledDesc*> &thread_layout)

Perform optimized blockwise copy between two tensors. Tensors must have the same size.

Note

At now Vgpr and Sgpr are not supported.

Template Parameters

- **DimAccessOrderTuple** – Tuple with dimension access order.
- **VectorDim** – Dimension for vectorize read and write.
- **ScalarPerVector** – Number of scalar per vectorize read and write.

Parameters

- **src_tensor** – Source tensor.
- **dst_tensor** – Destination tensor.
- **thread_layout** – Thread layout per each dimension for copy.

Functions

template<typename **DataType**, index_t **BlockSize**, typename **GemmTraits**, typename **ATensorType**, typename **BTensorType**, typename **CTensorType**>
__device__ void **blockwise_gemm_xdl**(const *ATensorType* &a_local_tile_tensor, const *BTensorType* &b_local_tile_tensor, *CTensorType* &c_reg_tensor)

Perform blockwise gemm xdl on tensors stored in lds. Result will be stored in Vgpr register. A data layout must be (MPerBlock, KPerBlock) or (K0PerBlock, MPerBlock, K1) and B data layout must be (NPerBlock, KPerBlock) or (K0PerBlock, NPerBlock, K1).

Note

C output Vgpr register layout (8D):

- MXdlPerWave - The number of MFMA instructions run by single wave in M dimension per tile.
- NXdlPerWave - The number of MFMA instructions run by single wave in N dimension per tile.
- MWave - Equals to 1 since this is for single wave.
- NWave - Equals to 1 since this is for single wave.
- NumGroupsPerBlock - Mfma instruction internal layout (depeneds on the instruction size).
- NumInputsBlock - Mfma instruction internal layout (depeneds on the instruction size).
- GroupSize - Mfma instruction internal layout (depeneds on the instruction size).
- NumThreadsPerBlock - Mfma instruction internal layout (depeneds on the instruction size).

Template Parameters

- **DataType** – Input data types.
- **BlockSize** – *Tensor* to pad.
- **GemmTraits** – Traits of gemm xdl operation.

Parameters

- **a_local_tile_tensor** – A tensor in LDS memory for blockwise gemm (MPerBlock, KPerBlock) or (K0PerBlock, MPerBlock, K1) layout.
- **b_local_tile_tensor** – B tensor in LDS memory for blockwise gemm (NPerBlock, KPerBlock) or (K0PerBlock, NPerBlock, K1) layout.
- **c_reg_tensor** – C tensor VGPR memory for blockwise gemm.

template<typename **DataType**, typename **ATileLayout**, typename **BTileLayout**, index_t **BlockSize**, typename **GemmTraits**, typename **CTensorType**>
__host__ __device__ constexpr auto **make_blockwise_gemm_xdl_c_local_partition**(*CTensorType* &c_local_tile_tensor)

Create local partition per thread for C tensor.

Note

C output global memory layout (8D):

- MXdlPerWave - The number of MFMA instructions run by single wave in M dimension.
- NXdlPerWave - The number of MFMA instructions run by single wave in N dimension.
- MWave - The number of waves in single tile M dimension per tile.
- NWave - The number of waves in single tile N dimension per tile.
- NumGroupsPerBlock - Mfma instruction internal layout (depeneds on the instruction size).
- NumInputsBlock - Mfma instruction internal layout (depeneds on the instruction size).
- GroupSize - Mfma instruction internal layout (depeneds on the instruction size).
- NumThreadsPerBlock - Mfma instruction internal layout (depeneds on the instruction size).

Template Parameters

- **DataType** – Input data types.
- **ATileLayout** – A tensor layout.
- **BTileLayout** – B tensor layout.
- **BlockSize** – Number of threads in block.
- **GemmTraits** – Traits of gemm xdl operation.

Parameters

c_local_tile_tensor – C tensor in LDS memory for blockwise gemm (MPerBlock, NPerBlock) layout.

Returns

Partition c tensor for blockwise gemm.

template<typename **DataType**, typename **ATileLayout**, typename **BTileLayout**, index_t **BlockSize**, typename **GemmTraits**>

__host__ __device__ constexpr auto **make_blockwise_gemm_xdl_c_vgpr**()

Create local partition per thread for C tensor.

> **Note**
>
> C output Vgpr register layout (8D):
>
> - MXdlPerWave - The number of MFMA instructions run by single wave in M dimension per tile.
> - NXdlPerWave - The number of MFMA instructions run by single wave in N dimension per tile.
> - MWave - Equals to 1 since this is for single wave.
> - NWave - Equals to 1 since this is for single wave.
> - NumGroupsPerBlock - Mfma instruction internal layout (depeneds on the instruction size).
> - NumInputsBlock - Mfma instruction internal layout (depeneds on the instruction size).
> - GroupSize - Mfma instruction internal layout (depeneds on the instruction size).
> - NumThreadsPerBlock - Mfma instruction internal layout (depeneds on the instruction size).

Template Parameters

- **DataType** – Input data types.
- **ATileLayout** – A tensor layout.
- **BTileLayout** – B tensor layout.
- **BlockSize** – Number of threads in block.
- **GemmTraits** – Traits of gemm xdl operation.

Returns

Vgpr c tensor for blockwise gemm.

CHAPTER

ELEVEN

CLASS LIST

CHAPTER

TWELVE

CONTRIBUTOR'S GUIDE

This chapter explains the rules for contributing to the Composable Kernel project, and how to contribute.

12.1 Getting started

1. **Documentation:** Before contributing to the library, familiarize yourself with the Composable Kernel User Guide. It provides insight into the core concepts, environment configuration, and steps to obtain or build the library. You can also find some of this information in the README file on the project's GitHub page. from the AMD Community portal. It offers a deeper understanding of the library's objectives and showcases its performance capabilities.
2. **General information:** For broader information about AMD products, consider exploring the AMD Developer Central portal.

12.2 How to contribute

You can make an impact by reporting issues or proposing code enhancements through pull requests.

12.2.1 Reporting issues

Use Github issues to track public bugs and enhancement requests.

If you encounter an issue with the library, please check if the problem has already been reported by searching existing issues on GitHub. If your issue seems unique, please submit a new issue. All reported issues must include:

- A comprehensive description of the problem, including:
  - What did you observe?
  - Why do you think it is a bug (if it seems like one)?
  - What did you expect to happen? What would indicate the resolution of the problem?
  - Are there any known workarounds?
- Your configuration details, including:
  - Which GPU are you using?
  - Which OS version are you on?
  - Which ROCm version are you using?
  - Are you using a Docker image? If so, which one?
- Steps to reproduce the issue, including:
  - What actions trigger the issue? What are the reproduction steps?

        * If you build the library from scratch, what CMake command did you use?
    - How frequently does this issue happen? Does it reproduce every time? Or is it a sporadic issue?

Before submitting any issue, ensure you have addressed all relevant questions from the checklist.

12.2.2 Creating Pull Requests

You can submit Pull Requests (PR) on GitHub.

All contributors are required to develop their changes on a separate branch and then create a pull request to merge their changes into the *develop* branch, which is the default development branch in the Composable Kernel project. All external contributors must use their own forks of the project to develop their changes.

When submitting a Pull Request you should:

- Describe the change providing information about the motivation for the change and a general description of all code modifications.
- Verify and test the change:
    - Run any relevant existing tests.
    - Write new tests if added functionality is not covered by current tests.
- Ensure your changes align with the coding style defined in the `.clang-format` file located in the project's root directory. We leverage *pre-commit* to run *clang-format* automatically. We highly recommend contributors utilize this method to maintain consistent code formatting. Instructions on setting up *pre-commit* can be found in the project's README file
- Link your PR to any related issues:
    - If there is an issue that is resolved by your change, please provide a link to the issue in the description of your pull request.
- For larger contributions, structure your change into a sequence of smaller, focused commits, each addressing a particular aspect or fix.

Following the above guidelines ensures a seamless review process and faster assistance from our end.

Thank you for your commitment to enhancing the Composable Kernel project!

CHAPTER

THIRTEEN

LICENSE

INDEX