MIGraphX

Release 2.15.0

Advanced Micro Devices, Inc.

May 21, 2026

INSTALLATION

1 MIGraphX on ROCm installation **3**
- 1.1 Install MIGraphX with a package installer . . . 3
- 1.2 Build MIGraphX from source . . . 3

2 Torch-MIGraphX installation **5**
- 2.1 Install Torch-MIGraphX . . . 5
- 2.2 Build Torch-MIGraphX from source . . . 5

3 Deep learning compilation with MIGraphX **7**
- 3.1 What MIGraphX provides . . . 9

4 MIGraphX user reference **11**
- 4.1 MIGraphX C++ API reference . . . 11
- 4.2 MIGraphX Python API reference . . . 19
- 4.3 Supported ONNX Operators . . . 26

5 MIGraphX contributor reference **37**
- 5.1 MIGraphX environment variables . . . 37
- 5.2 MIGraphX driver . . . 46
- 5.3 Developing for MIGraphX . . . 57
- 5.4 Data types . . . 59
- 5.5 Operators . . . 72
- 5.6 Program . . . 142
- 5.7 Targets . . . 146
- 5.8 Quantization . . . 148
- 5.9 Passes . . . 149
- 5.10 Matchers . . . 152
- 5.11 Tools . . . 153

6 MIGraphX examples **157**

7 License **159**

Python Module Index **161**

Index **163**

MIGraphX is a graph compiler and inference engine for high performance machine learning model inference. It compiles trained models from end-to-end to optimize for inference performance on AMD hardware.

The MIGraphX public repository is located at https://github.com/ROCm/AMDMIGraphX/.

You can integrate MIGraphX with PyTorch worflows by using the Torch-MIGraphX library. The public repository is located at https://github.com/ROCm/torch_migraphx/.

Install

- *MIGraphX on ROCm installation*
- *Torch-MIGraphX installation*

Conceptual

- *Deep learning compilation with MIGraphX*

Reference

- *MIGraphX user reference*
  - *MIGraphX C++ API reference*
  - *MIGraphX Python API reference*
  - *Supported ONNX Operators*
- *MIGraphX contributor reference*
  - *Environment variables*
  - *Develop for the MIGraphX code base*
  - *MIGraphX driver*

Examples

- *MIGraphX examples*

To contribute to the documentation refer to Contributing to ROCm.

Licensing information can be found on the Licensing page.

CHAPTER

ONE

MIGRAPHX ON ROCM INSTALLATION

ROCm must be installed before installing MIGraphX. See ROCm installation for Linux for instructions.

Installing MIGraphX using a package installer is sufficient for most users who want to use the MIGraphX API. If you plan to develop for MIGraphX or contribute to the source code, see Developing for MIGraphX

1.1 Install MIGraphX with a package installer

The package installer will install all the prerequisites you need for MIGraphX.

Use the following command to install MIGraphX:

```
sudo apt update && sudo apt install -y migraphx
```

1.2 Build MIGraphX from source

Note

This method for building MIGraphX requires using `sudo`.

1. Install `rocm-cmake`, `pip3`, `rocblas`, and `miopen-hip`:

```
sudo apt install -y rocm-cmake python3-pip rocblas miopen-hip
```

2. Install rbuild:

```
pip3 install --prefix /usr/local https://github.com/RadeonOpenCompute/rbuild/archive/
↪master.tar.gz
```

3. Build MIGraphX source code:

```
sudo rbuild build -d depend -B build -DGPU_TARGETS=$(/opt/rocm/bin/rocminfo | grep -o -
↪m1 'gfx.*')
```

CHAPTER

TWO

TORCH-MIGRAPHX INSTALLATION

MIGraphX can be integrated with PyTorch workflows by using the Torch-MIGraphX library. It includes the `torch.compile` API, so you can compile PyTorch models using MIGraphX.

Prerequisites:

- ROCm must be installed before installing MIGraphX. See ROCm installation for Linux for instructions.
- Installing MIGraphX using a package installer is sufficient if you want to use the MIGraphX API. See MIGraphX installation for instructions.
- Pytorch must be installed. See PyTorch installation for instructions.

2.1 Install Torch-MIGraphX

Use the following command to install `torch_migraphx` using `pip`:

```
pip install torch_migraphx
```

2.2 Build Torch-MIGraphX from source

Use the following command to build `torch_migraphx` from source:

```
git clone https://github.com/ROCm/torch_migraphx.git
cd torch_migraphx
pip install . --no-build-isolation
```

Refer to the https://github.com/ROCm/torch_migraphx/ repository for more information on local builds and Docker environments.

CHAPTER

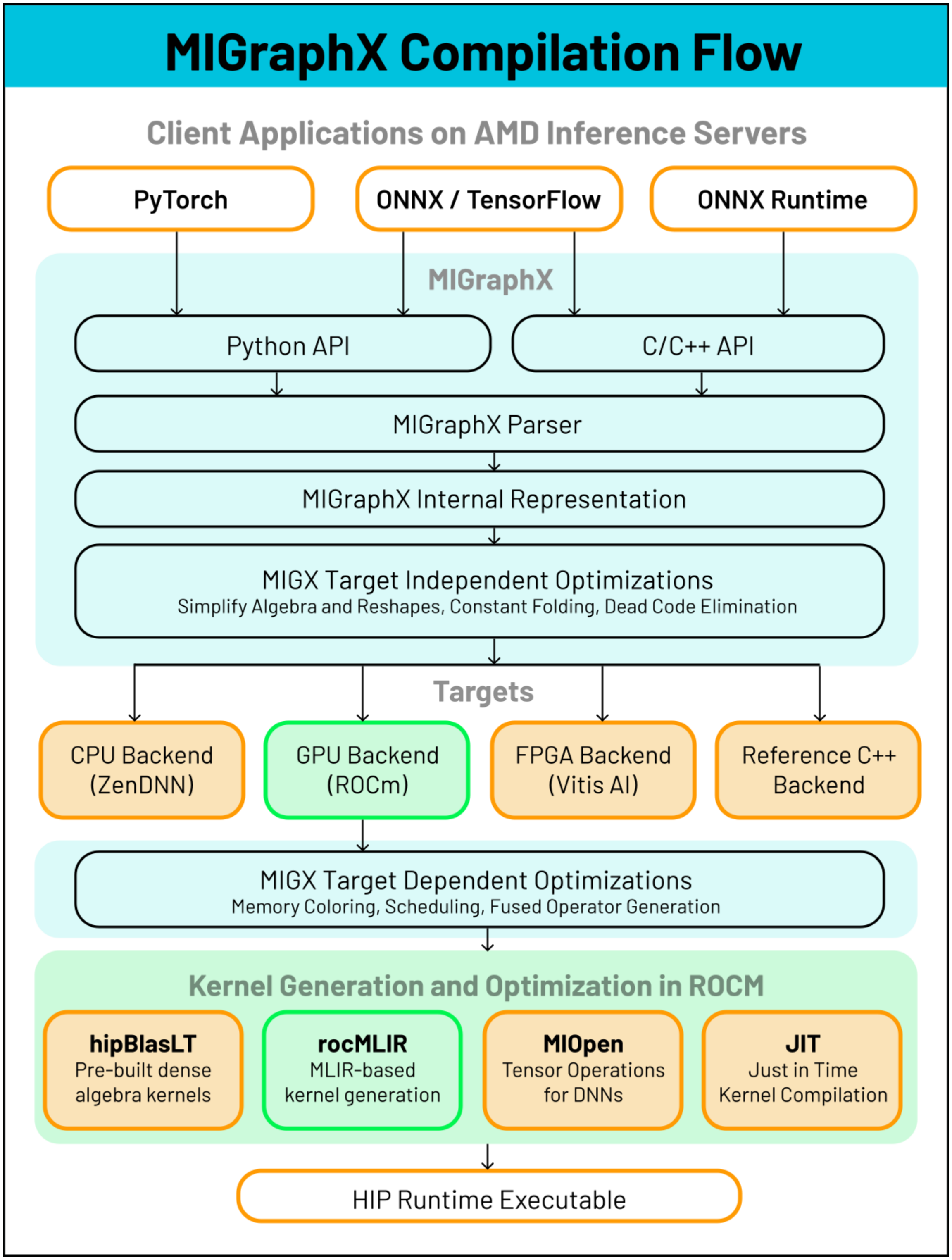
THREE

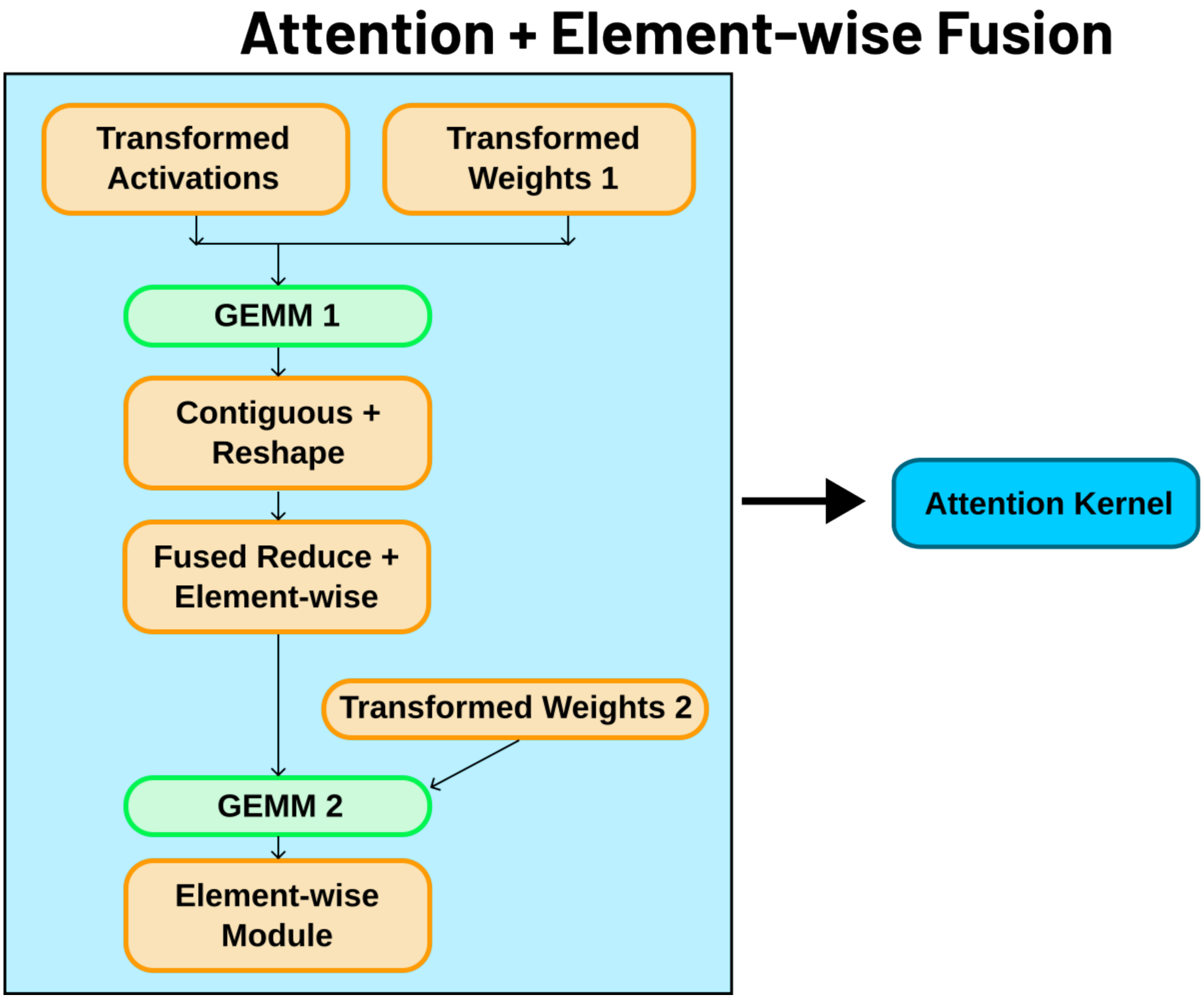
DEEP LEARNING COMPILATION WITH MIGRAPHX

The MIGraphX deep learning (DL) compiler improves inference by analyzing a model's compute graph and applying program transformations. After optimization, the compiler lowers graph operations to device kernels (from libraries or via code generation) for efficient execution.

A common transformation is kernel fusion; where compatible operations are merged into a single kernel launch. Fusion reduces launch overhead and avoids extra reads or writes between host and device, which typically improves latency and throughput. By applying graph-level optimizations and choosing or generating efficient device kernels, MIGraphX delivers high-performance over uncompiled models and less optimized compiled solutions.

An overview of the compilation process for MIGraphX is shown below in `compilation-label`. One type of optimization that MIGraphX performs are kernel fusions such as the Attention fusion seen in `attention-label`.


MIGraphX Compilation Flow
Client Applications on AMD Inference Servers
PyTorch
ONNX / TensorFlow
ONNX Runtime
MIGraphX
Python API
C/C++ API
MIGraphX Parser
MIGraphX Internal Representation
MIGX Target Independent Optimizations
Simplify Algebra and Reshapes, Constant Folding, Dead Code Elimination
Targets
CPU Backend (ZenDNN)
GPU Backend (ROCm)
FPGA Backend (Vitis AI)
Reference C++ Backend
MIGX Target Dependent Optimizations
Memory Coloring, Scheduling, Fused Operator Generation
Kernel Generation and Optimization in ROCM
hipBlasLT
Pre-built dense algebra kernels
rocMLIR
MLIR-based kernel generation
MIOpen
Tensor Operations for DNNs
JIT
Just in Time Kernel Compilation
HIP Runtime Executable



3.1 What MIGraphX provides

- **End-to-end:** compilation and execution of DL models on AMD GPUs
- **C++ implementation:** with Python and C++ APIs
- **Model inputs:**
  - ONNX and TensorFlow
  - PyTorch through https://github.com/ROCm/torch_migraphx
  - ONNX Runtime execution provider
- **Hardware targets:** AMD Navi (consumer) and MI (server) GPUs
- **Supported data types:** FP16, BF16, OCP FP8, INT8, INT4

CHAPTER

FOUR

MIGRAPHX USER REFERENCE

The MIGraphX library includes *C++* and *Python* APIs for creating applications.

To contribute to the MIGraphX code base, see *the MIGraphX contributor documentation*.

4.1 MIGraphX C++ API reference

4.1.1 shape

enum **migraphx_shape_datatype_t**

An enum to represent the different data type inputs.

Values:

enumerator **migraphx_shape_tuple_type**

enumerator **migraphx_shape_fp4x2_type**

enumerator **migraphx_shape_bool_type**

enumerator **migraphx_shape_half_type**

enumerator **migraphx_shape_float_type**

enumerator **migraphx_shape_double_type**

enumerator **migraphx_shape_uint8_type**

enumerator **migraphx_shape_int8_type**

enumerator **migraphx_shape_uint16_type**

enumerator **migraphx_shape_int16_type**

enumerator **migraphx_shape_int32_type** enumerator **migraphx_shape_int64_type**

enumerator **migraphx_shape_uint32_type**

enumerator **migraphx_shape_uint64_type**

enumerator **migraphx_shape_fp8e4m3fnuz_type**

enumerator **migraphx_shape_fp8e4m3fn_type**

enumerator **migraphx_shape_fp8e5m2_type**

enumerator **migraphx_shape_bf16_type**

enumerator **migraphx_shape_fp8e5m2fnuz_type**

template<class **Lens**, class **Strides**>

struct **shape** : public migraphx::handle_base<>, public migraphx::equality_comparable<*shape*<*Lens*, *Strides*>>

Describe shape of tensor.

A shape consists of a data type, lengths of multi-dimension tensor, and strides

Public Types

using **shape_type** = *shape*

using **index_array** = typename *Lens*::base_array

Public Functions

inline **shape**()

inline **shape**(const migraphx_shape *p)

template<class **HandleType**, class **Lifetime**, class = typename std::enable_if<std::is_convertible<*HandleType**, handle_type*>{}>::type>
inline **shape**(*HandleType* *p, *Lifetime* lifetime)

inline **shape**(*migraphx_shape_datatype_t* type)

Construct a scalar shape.

inline **shape**(*migraphx_shape_datatype_t* type, std::vector<size_t> plengths)

Construct a shape with its type and lengths. The strides are automatically computed assumming a packed layout.

inline **shape**(*migraphx_shape_datatype_t* t, std::initializer_list<std::size_t> d)

inline **shape**(*migraphx_shape_datatype_t* type, std::vector<size_t> plengths, std::vector<size_t> pstrides) inline **shape**(*migraphx_shape_datatype_t* type, const dynamic_dimensions &dyn_dims)

inline std::vector<size_t> **lengths**() const

inline std::vector<size_t> **strides**() const

inline dynamic_dimensions **dyn_dims**() const

Get the dynamic dimensions of the shape.

inline *migraphx_shape_datatype_t* **type**() const

inline size_t **elements**() const

inline size_t **bytes**() const

inline bool **standard**() const

inline bool **dynamic**() const

Is the shape dynamic.

inline size_t **index**(size_t i) const

constexpr **shape**() = default

inline constexpr **shape**(*Lens* l, *Strides* s)

inline constexpr auto **elements**() const

inline constexpr auto **element_space**() const

inline constexpr auto **packed**() const

inline constexpr auto **broadcasted**() const

inline constexpr auto **transposed**() const

inline constexpr auto **skips**() const

inline constexpr auto **standard**() const

inline constexpr index_int **index**(*index_array* x) const

inline constexpr index_int **index**(index_int i) const

inline constexpr index_int **compute_index**(index_int i) const

inline constexpr *index_array* **multi**(index_int idx) const

Convert single index into a multi-index.

inline constexpr index_int **single**(*index_array* idx) const

Convert multi-index into a single index.

inline constexpr *shape* **get_shape**() const

Public Members

Lens **lens** = { }

Strides **strides** = { }

Friends

inline friend bool **operator==**(const *shape* &px, const *shape* &py)

inline friend bool **operator!=**(const *shape* &px, const *shape* &py)

template<class ...**Ts**>
inline friend constexpr bool **operator==**(const *shape* &x, const *shape*<*Ts*...> &y)

template<class **Stream**>
inline friend constexpr const *Stream* &**operator<<**(const *Stream* &ss, const *shape* &s)

4.1.2 argument

struct **argument** : public migraphx::handle_base<>

Arguments to be passed to an migraphx arguments.

An `argument` represents a raw buffer of data with a shape.

Public Functions

inline **argument**()

template<class **HandleType**, class **Lifetime**, class = typename std::enable_if<std::is_convertible<*HandleType**, handle_type*>{}>::type>
inline **argument**(*HandleType* *p, *Lifetime* lifetime)

inline **argument**(const migraphx_argument *p)

inline **argument**(*shape* pshape)

inline **argument**(*shape* pshape, void *pbuffer)

inline *shape* **get_shape**() const

inline char ***data**() const

template<typename **T**>
inline std::vector<*T*> **as_vector**() const

Public Static Functions

static inline void **save_argument**(const *argument* &a, const std::string &filename)

Save an argument to a file.

static inline *argument* **load_argument**(const std::string &filename)

Load an argument from a file.

static inline *argument* **generate**(*shape* ps, size_t pseed = 0)

Generate an argument using random data.

Friends

inline friend bool **operator==**(const *argument* &px, const *argument* &py)

inline friend bool **operator!=**(const *argument* &px, const *argument* &py)

4.1.3 target

struct **target** : public migraphx::handle_base<>

A target for compilation.

Public Functions

inline **target**()

template<class **HandleType**, class **Lifetime**, class = typename std::enable_if<std::is_convertible<*HandleType**, handle_type*>{}>::type>
inline **target**(*HandleType* *p, *Lifetime* lifetime)

inline **target**(const char *name)

Construct a target from its name.

4.1.4 program

struct **program_parameter_shapes** : public migraphx::handle_base<>

Public Functions

inline **program_parameter_shapes**()

template<class **HandleType**, class **Lifetime**, class = typename std::enable_if<std::is_convertible<*HandleType**, handle_type*>{}>::type>
inline **program_parameter_shapes**(*HandleType* *p, *Lifetime* lifetime)

inline size_t **size**() const

inline *shape* **operator[]**(const char *pname) const

inline std::vector<const char*> **names**() const

struct **program_parameters** : public migraphx::handle_base<>

A class to construct the inputs parameters for a program.

Public Functions

template<class **HandleType**, class **Lifetime**, class = typename std::enable_if<std::is_convertible<*HandleType**, handle_type*>{}>::type>
inline **program_parameters**(*HandleType* *p, *Lifetime* lifetime)

inline **program_parameters**(migraphx_program_parameters *p)

inline **program_parameters**()

inline **program_parameters**(std::initializer_list<std::pair<std::string, *argument*>> l)

Construct the parameters from initializer_list.

inline void **add**(const char *pname, const *argument* &pargument) const

Add a new parameter.

struct **migraphx_compile_options**

Public Functions

template<class ...**Ts**>
inline **migraphx_compile_options**(*Ts*&&... xs)

Public Members

migraphx::compile_options **object**

struct **program** : public migraphx::handle_base<>

A program represents the all computation graphs to be compiled and executed.

Public Functions

inline **program**()

template<class **HandleType**, class **Lifetime**, class = typename std::enable_if<std::is_convertible<*HandleType**, handle_type*>{}>::type>
inline **program**(*HandleType* *p, *Lifetime* lifetime)

inline void **compile**(const *target* &ptarget, const compile_options &poptions) const

Compile the program for a specific target to be ran on.

inline void **compile**(const *target* &ptarget) const

Compile the program for a specific target to be ran on.

inline *program_parameter_shapes* **get_parameter_shapes**() const

Return the shapes for the input parameters.

inline shapes **get_output_shapes**() const

Get the shapes of all the outputs returned by this program.

inline arguments **eval**(const *program_parameters* &pparams) const

Run the program using the inputs passed in.

template<class **Stream**>
inline arguments **run_async**(const *program_parameters* &pparams, *Stream* *s) const

Overloaded to allow for execution_environment input.

inline void **print**() const

inline *program* **sort**()

call &**migraphx_program_get_main_module**()

inline context **experimental_get_context**()

call & migraphx_program_create_module (), name.data()

Public Members

return module = {p_modu, this->share_handle()}

Friends

inline friend bool **operator==**(const *program* &px, const *program* &py)

inline friend bool **operator!=**(const *program* &px, const *program* &py)

4.1.5 quantize

struct **quantize_op_names** : public migraphx::handle_base<>

Public Functions

inline **quantize_op_names**()

template<class **HandleType**, class **Lifetime**, class = typename std::enable_if<std::is_convertible<*HandleType**, handle_type*>{}>::type>
inline **quantize_op_names**(*HandleType* *p, *Lifetime* lifetime)

inline void **add**(const std::string &name)

inline void migraphx::**quantize_fp16**(const *program* &prog)

Quantize program to use fp16.

inline void migraphx::**quantize_fp16**(const *program* &prog, const *quantize_op_names* &names)

Quantize program to use fp16.

inline void migraphx::**quantize_bf16**(const *program* &prog)

Quantize program to use fp16.

struct **quantize_int8_options** : public migraphx::handle_base<>

Options to be passed when quantizing for int8.

Public Functions

inline **quantize_int8_options**()

template<class **HandleType**, class **Lifetime**, class = typename std::enable_if<std::is_convertible<*HandleType**, handle_type*>{}>::type>
inline **quantize_int8_options**(*HandleType* *p, *Lifetime* lifetime)

inline void **add_op_name**(const std::string &name)

Add an operator that should be quantized.

inline void **add_calibration_data**(const *program_parameters* &pp)

Add calibrartion data to be used for quantizing.

Public Members

std::vector<parameter_map> **calibration** = {}

std::unordered_set<std::string> **op_names** = {}

4.1.6 parse_onnx

struct **onnx_options** : public migraphx::handle_base<>

Options for parsing onnx options.

Public Functions

inline **onnx_options**()

template<class **HandleType**, class **Lifetime**, class = typename std::enable_if<std::is_convertible<*HandleType**, handle_type*>{}>::type>
inline **onnx_options**(*HandleType* *p, *Lifetime* lifetime)

inline void **set_input_parameter_shape**(const std::string &name, std::vector<std::size_t> dim)

Make onnx parser treat an inputs with a certain dimensions.

inline void **set_dyn_input_parameter_shape**(const std::string &name, const dynamic_dimensions &dyn_dims)

inline void **set_default_dim_value**(unsigned int value)

When there is a dimension parameter, then use this default value.

inline void **set_default_dyn_dim_value**(const dynamic_dimension &dd)

inline void **set_default_loop_iterations**(int64_t value)

Set default max iteration number for the loop operator.

inline void **set_limit_loop_iterations**(int64_t value)

Set max iteration limit for the loop operator.

inline void **set_external_data_path**(const std::string &external_data_path)

Set absolute path for external data files.

inline *program* migraphx::**parse_onnx**(const char *filename)

Parse an onnx file into a migraphx program.

inline *program* migraphx::**parse_onnx**(const char *filename, const migraphx::*onnx_options* &options)

Parse an onnx file into a migraphx program.

inline *program* migraphx::**parse_onnx_buffer**(const std::string &buffer)

Parse a buffer of memory as an onnx file.

inline *program* migraphx::**parse_onnx_buffer**(const std::string &buffer, const migraphx::*onnx_options* &options)

Parse a buffer of memory as an onnx file.

inline *program* migraphx::**parse_onnx_buffer**(const void *data, size_t size)

Parse a buffer of memory as an onnx file.

inline *program* migraphx::**parse_onnx_buffer**(const void *data, size_t size, const migraphx::*onnx_options* &options)

Parse a buffer of memory as an onnx file.

4.1.7 load

struct **file_options** : public migraphx::handle_base<>

Public Functions

template<class **HandleType**, class **Lifetime**, class = typename std::enable_if<std::is_convertible<*HandleType**, handle_type*>{}>::type>
inline **file_options**(*HandleType* *p, *Lifetime* lifetime)

inline **file_options**()

inline void **set_file_format**(const char *format)

inline *program* migraphx::**load**(const char *filename)

Load a saved migraphx program from a file.

inline *program* migraphx::**load**(const char *filename, const *file_options* &options)

Load a saved migraphx program from a file.

4.1.8 save

inline void migraphx::**save**(const *program* &p, const char *filename)

Save a program to a file.

inline void migraphx::**save**(const *program* &p, const char *filename, const *file_options* &options)

Save a program to a file.

4.2 MIGraphX Python API reference

4.2.1 shape

class migraphx.**shape**(*type*, *lens*, *strides=None*, *dyn_dims*)

Describes the shape of a tensor. This includes size, layout, and data type. Use dyn_dims for a dynamic shape.

migraphx.**type**()

An integer that represents the type.

Return type
int

migraphx.**lens**()

A list of the lengths of the shape.

Return type
list[int]

migraphx.**strides**()

A list of the strides of the shape.

Return type
list[int]

`migraphx.elements()`

The number of elements in the shape.

Return type
int

`migraphx.dyn_dims()`

The dynamic dimensions of the shape.

Return type
list[*dynamic_dimension*]

`migraphx.bytes()`

The number of bytes the shape uses.

Return type
int

`migraphx.type_size()`

The number of bytes one element uses

Return type
int

`migraphx.ndim()`

The number of dimensions for the shape.

Return type
int

`migraphx.packed()`

Returns true if the shape is packed.

Return type
bool

`migraphx.transposed()`

Returns true if the shape is transposed.

Return type
bool

`migraphx.broadcasted()`

Returns true if the shape is broadcasted.

Return type
bool

`migraphx.dynamic()`

Returns true if the shape is dynamic.

Return type
bool

`migraphx.standard()`

Returns true if the shape is a standard shape. That is, the shape is both packed and not transposed.

Return type
bool

`migraphx.scalar()`

Returns true if all strides are equal to 0 (scalar tensor).

Return type
bool

4.2.2 dynamic_dimension

`class migraphx.dynamic_dimension(min, max, optimals)`

Constructs a *dynamic_dimension* from a minimum, a maximum, and optionally a set of optimals.

`migraphx.is_fixed()`

Returns true if the *dynamic_dimension* is fixed.

:rtype : int

4.2.3 argument

`class migraphx.argument(data)`

Constructs an argument from a python buffer. This can include numpy arrays.

`migraphx.data_ptr()`

Returns the address to the underlying argument data.

Return type
int

`migraphx.get_shape()`

Returns the shape of the argument.

Return type
shape

`migraphx.tolist()`

Converts the elements of the argument to a python list.

Return type
list

`migraphx.generate_argument(s, seed=0)`

Generates an argument with random data.

Parameters

- **s** (shape) – Shape of argument to generate.
- **seed** (*int*) – The seed used for random number generation.

Return type
argument

`migraphx.fill_argument(s, value)`

Fills argument of shape *s* with the given value.

Parameters

- **s** (shape) – Shape of argument to fill.
- **value** (*int*) – Value to fill in the argument.

Return type
argument

`migraphx.create_argument`(*s*, *values*)

Creates an argument of shape *s* with a set of values.

Parameters

- **s** (`shape`) – Shape of argument to create.
- **values** (*list*) – Values to put in the argument. Must be the same number of elements as the shape.

Return type
argument

`migraphx.argument_from_pointer`(*shape*, *address*)

Creates argument from data stored in given address without copy.

Parameters

- **shape** (`shape`) – Shape of the data stored in address.
- **address** (*long*) – Memory address of data from another source

Return type
argument

`static argument.save`(*arg*, *filename*)

Saves argument to a file encoded in msgpack format.

Parameters

- **arg** (`argument`) – Argument to save out to.
- **filename** (*str*) – Path of file to save out to.

`static argument.load`(*filename*)

Load argument from a file encoded in msgpack format.

Parameters
filename (*str*) – Path of file to load.

Return type
argument

4.2.4 target

`class migraphx.target`

This represents the compilation target.

`migraphx.get_target`(*name*)

Constructs the target.

Parameters
name (*str*) – The name of the target to construct. This can either be 'gpu' or 'ref'.

Return type
target

4.2.5 module

`migraphx.print()`

Prints the contents of the module as list of instructions.

`migraphx.add_instruction(op, args, mod_args=[])`

Adds instruction into the module.

Parameters

- **op** (*operation*) – 'migraphx.op' to be added as instruction.
- **args** (*list[instruction]*) – list of inputs to the op.
- **mod_args** (*list[module]*) – optional list of module arguments to the operator.

:rtype instruction

`migraphx.add_literal(data)`

Adds constant or literal data of provided shape into the module from python buffer which includes numpy array.

Parameters

data (*py::buffer*) – Python buffer or numpy array

:rtype instruction

`migraphx.add_parameter(name, shape)`

Adds a parameter to the module with the provided name and shape.

Parameters

- **name** (*str*) – name of the parameter.
- **shape** (*shape*) – shape of the parameter.

:rtype instruction

`migraphx.add_return(args)`

Adds a return instruction into the module.

Parameters

args (*list[instruction]*) – instruction arguments which need to be returned from the module.

:rtype instruction

4.2.6 program

`class migraphx.program`

Represents the computation graph to be compiled and run.

`migraphx.clone()`

Makes a copy of the program.

Return type

program

`migraphx.get_parameter_names()`

Gets all the input argument's or parameter's names to the program as a list.

:rtype list[str]

migraphx.**get_parameter_shapes**()

Gets the shapes of all the input parameters in the program.

Return type

dict[str, *shape*]

migraphx.**get_output_shapes**()

Gets the shapes of the final outputs of the program.

Return type

list[*shape*]

migraphx.**compile**(*t*, *offload_copy=True*, *fast_math=True*, *exhaustive_tune=False*)

Compiles the program for the target and optimizes it.

Parameters

- **t** (target) – Compilation target for the program.
- **offload_copy** (*bool*) – For targets with offloaded memory(such as the gpu), this will insert instructions during compilation to copy the input parameters to the offloaded memory and to copy the final result from the offloaded memory back to main memory.
- **fast_math** (*bool*) – Optimize math functions to use faster approximate versions. There may be slight accuracy degredation when enabled.
- **exhaustive_tune** – Flag to enable exhaustive search to find the fastest version of generated kernels for selected backend.

migraphx.**get_main_module**()

Gets main module of the program.

:rtype module

migraphx.**create_module**(*name*)

Creates and adds a module with the provided name into the program.

:param str name : name of the new module. :rtype module

migraphx.**run**(*params*)

Runs the program.

Parameters

params (*dict[str, argument]*) – Map of the input parameters to be used when running the program.

Returns

The result of the last instruction.

Return type

list[*argument*]

migraphx.**sort**()

Sorts the modules of the program for the instructions to appear in topologically sorted order.

migraphx.**quantize_fp16**(*prog*, *ins_names=['all']*)

Quantizes the program to use fp16.

Parameters

- **prog** (program) – Program to quantize.
- **ins_names** (*list[str]*) – List of instructions to quantize.

`migraphx.quantize_bf16(prog, ins_names=['all'])`

Quantizes the program to use bf16.

Parameters

- **prog** (program) – Program to quantize.
- **ins_names** (*list[str]*) – List of instructions to quantize.

`migraphx.quantize_int8(prog, t, calibration=[], ins_names=['dot', 'convolution'])`

Quantizes the program to use int8.

Parameters

- **prog** (program) – Program to quantize.
- **t** (target) – Target to be used to run the calibration data.
- **calibration** (*list[dict[str, argument]]*) – Calibration data used to decide the parameters to the int8 optimization.
- **ins_names** (*list[str]*) – List of instructions to quantize.

`migraphx.autocast_fp8(prog)`

Auto-convert FP8 parameters and return values to Float for an MIGraphX program.

Parameters

prog (program) – Program to auto-convert parameters/return values.

4.2.7 op

4.2.8 parse_onnx

`migraphx.parse_onnx(filename, default_dim_value=1, map_input_dims={}, skip_unknown_operators=false, print_program_on_error=false, max_loop_iterations=10, limit_max_iterations=65535)`

Loads and parses an ONNX file.

Parameters

- **filename** (*str*) – Path to file.
- **default_dim_value** (*str*) – default dimension to use (if not specified in onnx file).
- **default_dyn_dim_value** (dynamic_dimension) – default dynamic_dimension value to use.
- **map_input_dims** (*str*) – Explicitly specify the dims of an input.
- **map_dyn_input_dims** (*list[*dynamic_dimension*]*) – Explicitly specify the dynamic_dimensions of an input.
- **skip_unknown_operators** (*str*) – Continue parsing onnx file if an unknown operator is found.
- **print_program_on_error** (*str*) – Print program if an error occurs.
- **max_loop_iterations** (*int*) – Maximum iteration number for the loop operator if trip count is not set.
- **limit_max_iterations** (*int*) – Maximum iteration limit for the loop operator.

Return type

program

4.2.9 parse_tf

`migraphx.parse_tf`(*filename*, *is_nhwc=True*, *batch_size=1*, *map_input_dims=dict()*, *output_names=[]*)

Loads and parses a tensorflow protobuf file.

Parameters

- **filename** (*str*) – Path to file.
- **is_nhwc** (*bool*) – Use nhwc as default format.
- **batch_size** (*str*) – default batch size to use (if not specified in protobuf).
- **map_input_dims** (*dict[str, list[int]]*) – Optional arg to explictly specify dimensions of the inputs.
- **output_names** (*list[str]*) – Optional argument specify names of the output nodes.

Return type

program

4.2.10 load

`migraphx.load`(*filename*, *format='msgpack'*)

Loads a MIGraphX program.

Parameters

- **filename** (*str*) – Path to file.
- **format** (*str*) – Format of file. Valid options are msgpack or json.

Return type

program

4.2.11 save

`migraphx.save`(*p*, *filename*, *format='msgpack'*)

Saves a MIGraphX program.

Parameters

- **p** (`program`) – Program to save.
- **filename** (*str*) – Path to file.
- **format** (*str*) – Format of file. Valid options are msgpack or json.

4.3 Supported ONNX Operators

MIGraphX supports operators up to Opset 19. Latest information of ONNX operators can be found in the ONNX GitHub repository.

MIGraphX supports the following ONNX data types: BOOL, UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FLOAT8, FLOAT16, FLOAT32, and DOUBLE

> **Note**
>
> FP8 support is only for E4M3FNUZ, see Float stored in 8 bits in the ONNX documentation.

See below for the support matrix of ONNX operators in MIGraphX.

> **Note**
>
> The listed supported types are taken from the ONNX specification. An operator might support other additional datatypes.

4.3.1 Operator Support Matrix

Operator	Supported	Supported Types	Limitations
Abs		UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	
Acos		FP8, FP16, FP32, FP64	
Acosh		FP8, FP16, FP32, FP64	
Add		UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	
And		BOOL	
ArgMax		UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	
ArgMin		UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	
Asin		FP8, FP16, FP32, FP64	
Asinh		FP8, FP16, FP32, FP64	
Atan		FP8, FP16, FP32, FP64	
Atanh		FP8, FP16, FP32, FP64	
AveragePool		FP8, FP16, FP32, FP64	
BatchNormalization		FP8, FP16, FP32, FP64	
BiasGelu		FP8, FP16, FP32, FP64	
Bernoulli			
BitShift			
BitwiseAnd			
BitwiseNot			
BitwiseOr			
BitwiseXor			
BlackmanWindow			
Cast		BOOL, UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP16, FP32, FP64	`saturate` not supported

continues on next page

Table 1 – continued from previous page

Operator	Supported	Supported Types	Limitations
CastLike		BOOL, UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP16, FP32, FP64	`saturate` not supported
Ceil		FP8, FP16, FP32, FP64	
Celu		FP32	
CenterCropPad			
Clip		UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	
Col2Im			
Compress			
Concat		BOOL, UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	
ConcatFromSequence			
Constant		BOOL, UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	
ConstantOfShape		BOOL, UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP16, FP32, FP64	dynamic shape is not supported
Conv		FP8, FP16, FP32, FP64	
ConvInteger		INT8	
ConvTranspose		FP8, FP16, FP32, FP64	`auto padding` not supported, `output_padding` math differs
Cos		FP8, FP16, FP32, FP64	
Cosh		FP8, FP16, FP32, FP64	
CumSum		UINT32, UINT64, INT32, INT64, FP8, FP16, FP32, FP64	`axis` dynamic shape is not supported
DFT			
DeformConv			
DepthToSpace		BOOL, UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	
DequantizeLinear		UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	
Det			

continues on next page

Table 1 – continued from previous page

Operator	Supported	Supported Types	Limitations
Div		UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	
Dropout		Any	Changed to `identity`
DynamicQuantizeLinear		FP8, FP16, FP32, FP64	`uint8` quantization only, dynamic shape is not supported
Einsum		Any	more than 1 diagonal per input is not supported e.g. `iijj->ij` batch diagonal where batches are not the leading dims is not supported e.g. `ii...->i...`
Elu		FP8, FP16, FP32, FP64	
Equal		BOOL, UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	
Erf		UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	
Exp		FP8, FP16, FP32, FP64	
Expand		BOOL, UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	dynamic shape is not supported
EyeLike		BOOL, UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	
FastGelu		FP8, FP16, FP32	
Flatten		BOOL, UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	
Floor		FP8, FP16, FP32, FP64	
Gather		BOOL, UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	
GatherElements		BOOL, UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	

continues on next page

Table 1 – continued from previous page

Operator	Supported	Supported Types	Limitations
GatherND		BOOL, UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	
Gelu		FP8, FP16, FP32, FP64	
Gemm		UINT32, UINT64, INT32, INT64, FP8, FP16, FP32, FP64	
GlobalAveragePool		FP8, FP16, FP32, FP64	
GlobalLpPool		FP8, FP16, FP32, FP64	
GlobalMaxPool		FP8, FP16, FP32, FP64	
Greater		UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	
GreaterOrEqual		UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	
GridSample		UINT32, UINT64, INT32, INT64, FP16, FP32, FP64	*5-D inputs* not supported
GroupNormalization		FP8, FP16, FP32, FP64	`stash_type` not supported
GRU		FP8, FP16, FP32, FP64	`Affine`, `ThresholdRelu`, `ScaledTanh`, `HardSigmoid`, `Softsign`, `Softplus` optional activation functions are not enabled
HammingWindow			
HannWindow			
HardSigmoid		FP8, FP16, FP32, FP64	
HardSwish		FP8, FP16, FP32, FP64	
Hardmax		FP8, FP16, FP32, FP64	
Identity		BOOL, UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	`identity`, `sequence` datatypes are not supported
If		BOOL, UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	`identity`, `sequence` datatypes are not supported
InstanceNormalization		FP16, FP32, FP64	
IsInf		FP8, FP16, FP32, FP64	
IsNaN		FP8, FP16, FP32, FP64	

continues on next page

Table 1 – continued from previous page

Operator	Supported	Supported Types	Limitations
LayerNormalization		FP8, FP16, FP32, FP64	`stash_type` not supported
LeakyRelu		FP8, FP16, FP32, FP64	
Less		UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	
LessOrEqual		UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	
Log		FP8, FP16, FP32, FP64	
LogSoftmax		FP8, FP16, FP32, FP64	
Loop		UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	`identity`, `sequence` datatypes are not supported, `max_iteration` has upper-bound
LRN		FP8, FP16, FP32, FP64	
LSTM		FP32, FP16	`Affine`, `ThresholdRelu`, `ScaledTanh`, `HardSigmoid`, `Softsign`, `Softplus` optional activation functions are not enabled
LpNormalization		FP8, FP16, FP32, FP64	
LpPool		FP32, FP16, FP8, INT8	`lpnorm` not supported pooling mode on GPU (MIOpen limitation)
MatMul		UINT32, UINT64, INT32, INT64, FP8, FP16, FP32, FP64	
MatMulInteger		UINT8, INT8	dynamic shape is not supported
Max		UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	
MaxPool		FP32, FP16, FP8, INT8	`storage_order` not supported, `dialtion` is partially supported on GPU (MIOpen limitation), `indices` 2nd ouput not supported
MaxRoiPool			
MaxUnpool			

continues on next page

Table 1 – continued from previous page

Operator	Supported	Supported Types	Limitations
Mean		FP8, FP16, FP32, FP64	
MeanVarian ceNormalization		FP8, FP16, FP32, FP64	
MelWeightMatrix			
Min		UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	
Mish			
Mod		UINT8, UINT16, UINT32, UINT64, INT16, INT64, FP8, FP16, FP32, FP64	`int8`, `int32` issue on GPU
Mul		UINT8, UINT16, UINT32, UINT64, INT16, INT64, FP8, FP16, FP32, FP64	
Multinomial		FP8, FP16, FP32, FP64	
Neg		INT8, INT32, INT64, FP8, FP16, FP32, FP64	
NegativeLogLikelihood-Loss			
NonMaxSuppression		FP8, FP16, FP32, FP64	fixed output size unless `use_dyn_output` set
NonZero		FP8, FP16, FP32, FP64	fixed output size unless `use_dyn_output` set
Not		BOOL	
OneHot		UINT8, UINT16, UINT32, UINT64, INT16, INT64, FP8, FP16, FP32, FP64	dynamic shape is not supported
Optional			
OptionalGetElement			
OptionalHasElement			
Or		BOOL	
Pad		BOOL, UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	`edge`, `warp` padding not supported, `pads` must be constant
Pow		UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	
PRelu		UINT32, UINT64, INT32, INT64, FP8, FP16, FP32, FP64	
QLinearAdd		UINT8, INT8	
QLinearAveragePool		UINT8, INT8	

continues on next page

Table 1 – continued from previous page

Operator	Supported	Supported Types	Limitations
QLinearConcat		UINT8, INT8	
QLinearConv		UINT8, INT8	
QLinearGlobalAverage-Pool		UINT8, INT8	
QLinearLeakyRelu		UINT8, INT8	
QLinearMatMul		UINT8, INT8	non-scalar inputs are not supported
QLinearMul		UINT8, INT8	
QLinearSigmoid		UINT8, INT8	
QuantizeLinear		FP8, FP16, FP32, INT32	`saturate` is not supported
RandomNormal		FP16, FP32, FP64	
RandomNormalLike		FP16, FP32, FP64	
RandomUniform		FP16, FP32, FP64	
RandomUniformLike		FP16, FP32, FP64	
Range		FP16, FP32, FP64, INT16, INT32, INT64	`start`, `end`, `delta` dynamic shape is not supported
Reciprocal		FP8, FP16, FP32, FP64	
ReduceL1		FP16, FP32, FP64, UINT32, UINT64, INT32, INT64	`axes` dynamic shape is not supported
ReduceL2		FP16, FP32, FP64, UINT32, UINT64, INT32, INT64	`axes` dynamic shape is not supported
ReduceLogSum		FP16, FP32, FP64, UINT32, UINT64, INT32, INT64	`axes` dynamic shape is not supported
ReduceLogSumExp		FP16, FP32, FP64, UINT32, UINT64, INT32, INT64	`axes` dynamic shape is not supported
ReduceMax		FP16, FP32, FP64, UINT32, UINT64, INT32, INT64	`axes` dynamic shape is not supported
ReduceMean		FP16, FP32, FP64, UINT32, UINT64, INT32, INT64	`axes` dynamic shape is not supported
ReduceMin		FP16, FP32, FP64, UINT32, UINT64, INT32, INT64	`axes` dynamic shape is not supported
ReduceProd		FP16, FP32, FP64, UINT32, UINT64, INT32, INT64	`axes` dynamic shape is not supported
ReduceSum		FP16, FP32, FP64, UINT32, UINT64, INT32, INT64	`axes` dynamic shape is not supported
ReduceSumSquare		FP16, FP32, FP64, UINT32, UINT64, INT32, INT64	`axes` dynamic shape is not supported

continues on next page

Table 1 – continued from previous page

Operator	Supported	Supported Types	Limitations
Relu		FP16, FP32, FP64, INT8, INT16, INT32, INT64	
Reshape		FP32, FP16, INT32, INT64, FP8, INT8, BOOL	`allowzero` not supported, dynamic shape is not supported
Resize		UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	`cubic`, `half_pixel_symmetric`, `tf_crop_and_resize` not supported, `linear` mode not supported for non-constant inputs, `exclude_outside` 1 is not supported, `antialias`, `extrapolation_value`, `keep_aspect_ratio_policy` not supported
ReverseSequence		BOOL, UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	variable `sequence_lens` is not supported
RNN		FP32, FP16	`Affine`, `ThresholdRelu`, `ScaledTanh`, `HardSigmoid`, `Softsign`, `Softplus` optional activation functions are not enabled
RoiAlign		FP8, FP16, FP32, FP64	
Round		FP8, FP16, FP32, FP64	
STFT			
Scan		UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	`identity`, `sequence` datatypes are not supported, Number of iterations has upper-bound Version 8 not supported
Scatter (deprecated)		BOOL, UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	
ScatterElements		BOOL, UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	
ScatterND		BOOL, UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	
Selu		FP8, FP16, FP32, FP64	
SequenceAt			

continues on next page

Table 1 – continued from previous page

Operator	Supported	Supported Types	Limitations
SequenceConstruct			
SequenceEmpty			
SequenceErase			
SequenceInsert			
SequenceLength			
SequenceMap			
Shape		BOOL, UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	
Shrink		UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	
Sigmoid		FP8, FP16, FP32, FP64	
Sign		UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	
Sin		FP8, FP16, FP32, FP64	
Sinh		FP8, FP16, FP32, FP64	
Size		BOOL, UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	
Slice		BOOL, UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	variable inputs are not supported
Softmax		FP8, FP16, FP32, FP64	
SoftmaxCrossEntropyLoss			
Softplus		FP8, FP16, FP32, FP64	
Softsign		FP8, FP16, FP32, FP64	
SpaceToDepth		BOOL, UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	
Split		BOOL, UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	dynamic shape is not supported
SplitToSequence			
Sqrt		FP8, FP16, FP32, FP64	
Squeeze		BOOL, UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	variable `axes` is not supported

continues on next page

Table 1 – continued from previous page

Operator	Supported	Supported Types	Limitations
StringNormalizer			
Sub		UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	
Sum		FP8, FP16, FP32, FP64	
Tan		FP8, FP16, FP32, FP64	
Tanh		FP8, FP16, FP32, FP64	
TfIdfVectorizer			
ThresholdedRelu		FP8, FP16, FP32, FP64	
Tile		BOOL, UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	dynamic shape is not supported
TopK		UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	dynamic `k` is not supported, `sorted` is not supported
Transpose		BOOL, UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	
Trilu		BOOL, UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	dynamic `k` is not supported
Unique		Any	only `axis = 0` is supported
Unsqueeze		BOOL, UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	variable `axes` is not supported
Upsample (deprecated)		BOOL, UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	
Where		BOOL, UINT8, UINT16, UINT32, UINT64, INT8, INT16, INT32, INT64, FP8, FP16, FP32, FP64	mixed static and dynamic shape inputs are not supported
Xor		BOOL	

CHAPTER

FIVE

MIGRAPHX CONTRIBUTOR REFERENCE

Developers who want to contribute to the MIGraphX code base can consult *Developing for MIGraphX*.

Developers who want to use the MIGraphX library in their own applications should see *the MIGraphX user reference* instead.

5.1 MIGraphX environment variables

The MIGraphX environment variables can be used by contributors to the MIGraphX code base to customize tuning, verification, and tracing.

5.1.1 Model performance tunable variables

Model performance tunable variables change the compilation behavior of a model. These are the most commonly used variables.

Environment variable	Values
`MIGRAPHX_ENABLE_NHWC` Forces the model to use the NHWC layout.	1: Forces the use of the NHWC layout. 0: Returns to default behavior. Default: The use of the NHWC layout isn't forced.
`MIGRAPHX_DISABLE_MLIR` When set, the rocMLIR library won't be used.	1: The rocMLIR library won't be used. 0: Returns to default behavior. Default: The rocMLIR library is used.
`MIGRAPHX_ENABLE_CK` When set, the Composable Kernel library is used.	Use with `MIGRAPHX_DISABLE_MLIR = 1`. 1: The Composable Kernel library is used. 0: Returns to default behavior. Default: Composable Kernel library isn't used.
`MIGRAPHX_SET_GEMM_PROVIDER` Sets the GEMM provider to be either rocBLAS or hipBLASlt.	`hipblaslt`: hipBLASLt is used as the GEMM provider. `rocblas`: rocBLAS is used as the GEMM provider. Default: `rocblas` on gfx90a; `hipblaslt` on all other architectures.
`MIGRAPHX_ENABLE_LAYERNORM_FUSION` When set, layernorm fusion is used.	1: Layernorm fusion will be used. 0: Returns to default behavior. Default: Layernorm fusion is not used.
`MIGRAPHX_DISABLE_MIOPEN_POOLING` When set, MIGraphX pooling is used instead of MIOpen pooling.	1: Use MIGraphX pooling. 0: Returns to default behavior. Default: MIOpen pooling is used.
`MIGRAPHX_USE_FAST_SOFTMAX` Turns on fast softmax optimization to speed up softmax computations.	1: Turns on Softmax optimization. 0: Returns to default behavior. Default: Softmax optimization is turned off.
`MIGRAPHX_DISABLE_FP32_SOFTMAX` Disables upcasting to fp32 when computing softmax for lower precision graphs.	1: Disables forcing full precision computation of softmax 0: Returns to default behavior. Default: Upcasting to FP32 is turned on.
`MIGRAPHX_MLIR_USE_SPECIFIC_OPS`	Takes a comma-separated list of operations. Operations can be any

5.1.2 Matching

Debug settings for matchers. Matchers are responsible for finding optimizations in the graph compilation stage.

Environment variable	Values
`MIGRAPHX_TRACE_MATCHES` When set, prints the name of matchers that have found a valid pattern match.	`1`: Prints the name of the matchers that have found a valid match. `2`: When used with `MIGRAPHX_TRACE_MATCHES_FOR`, prints the names of matchers that have been tried but which have not necessarily found a match. `0`: Returns to default behavior. Default: Nothing is printed.
`MIGRAPHX_TRACE_MATCHES_FOR` Turns on the printing of traces for the specified matcher if a string is found in the matcher's `file-name`, `function-name`, or `matcher-name`.	Takes a string to match.
`MIGRAPHX_VALIDATE_MATCHES` When set, `module.validate()` is used to validate the module after finding matches.	`1`: Runs `module.validate()`. `0`: Returns to default behavior. Default: `module.validate()` isn't run.
`MIGRAPHX_TIME_MATCHERS` When set, prints the time spent on a matcher. This helps identify time-consuming patterns.	``1`: Prints the time spent on the matcher. `0`: Returns to default behavior. Default: The time is not printed.

5.1.3 Pass controls

Debug settings for passes.

Environment variable	Values
MIGRAPHX_TRACE_ELIMINATE_C Turns on the printing of debug statements for `eliminate contiguous instruction` passes.	1: Debug statements are printed for `eliminate contiguous instructions` passes. 0: Returns to default behavior. Default: Debug statements aren't printed for `eliminate contiguous instructions` passes.
MIGRAPHX_DISABLE_POINTWISE When set, the `fuse_pointwise compile` pass isn't run.	1: The `fuse_pointwise compile` pass isn't run. 0: Returns to default behavior. Default: The `fuse_pointwise compile` pass is run.
MIGRAPHX_DEBUG_MEMORY_COLO Turns on the printing of debug statements for the `memory-coloring` pass.	1: Debug statements for the `memory-coloring` pass are printed. 0: Returns to default behavior. Default: Debug statements for the `memory-coloring` pass aren't printed.
MIGRAPHX_TRACE_SCHEDULE Turns on the printing of debug statements for the `schedule` pass.	1: Debug statements for the `schedule` pass are printed. 0: Returns to default behavior. Default: Debug statements for the `memory-coloring` pass aren't printed.
MIGRAPHX_TRACE_PROPAGATE_C Turns on tracing of instructions that have been replaced with a constant.	1: Instructions that have been replaced with a constant are traced. 0: Returns to default behavior. Default: Instructions that have been replaced with a constant aren't traced.
MIGRAPHX_DISABLE_DNNL_POST When set, the DNNL post-ops workaround isn't used.	1: The DNNL post-ops workaround ins't used. 0: Returns to default behavior. Default: The DNNL post-ops workaround is used.
MIGRAPHX_DISABLE_MIOPEN_FU When set, MIOpen fusions aren't used.	1: MIOpen fusions aren't used. 0: Returns to default behavior. Default: MIOpen fusions are used.
MIGRAPHX_DISABLE_SCHEDULE_ When set, the `schedule` pass isn't run.	1: The `schedule` pass isn't run. 0: Returns to default behavior. Default: The `schedule` pass is run.

5.1.4 Compilation tracing

Environment variable	Values
`MIGRAPHX_TRACE_FINALIZE` Turns on printing of graph instructions during the `module.finalize()` step.	`1`: Graph instructions will be printed. `0`: Returns to default behavior. Default: Graph instructions won't be printed.
`MIGRAPHX_TRACE_COMPILE` Turns on graph compilation tracing.	`1`: Turns on graph compilation tracing. `0`: Returns to default behavior. Default: Graph compilation isn't traced.
`MIGRAPHX_TRACE_ONNX_PARSEF` Turns on node-by-node tracing for the ONNX parser.	`1`: Node-by-node tracing is turned on. `0`: Returns to the default behavior. Default: There is no node-by-node tracing of the ONNX parser.
`MIGRAPHX_TRACE_EVAL` Turns on model evaluation tracing and sets its tracing level.	`1`: Print the run instructions and the time taken to complete the evaluation. `2`: Print the run instructions, time taken, a snippet of the output, and some statistics. `3`: Print the run instructions, time taken, a snippet of the output, and statistics for all output buffers.
`MIGRAPHX_TRACE_QUANTIZATIC` Turns on the printing of the traces for passes run during quantization.	`1`: Traces for passes run during quantization will be printed. `0`: Returns to default behavior. Default: The traces for passes run during quantization won't be printed out.
`MIGRAPHX_8BITS_QUANTIZATIC` Turns on the printing of the quantization parameters in the main module only.	`1`: Only the quantization parameters in the main module are printed. `0`: Returns to default behavior. Default:

5.1.5 MLIR

Environment variable	Values
`MIGRAPHX_TRACE_MLIR` Sets the MLIR trace level.	1: MLIR trace failures are printed. 2: MLIR trace failures are printed and all MLIR operations are printed as well.
`MIGRAPHX_MLIR_TUNING_DB` The path of the tuning database.	Takes the path to the tuning database.
`MIGRAPHX_MLIR_TUNING_CFG` Sets the path to the tuning configuration file to use with rocMLIR tuning scripts.	Takes the path to the configuration file. For example: `MIGRAPHX_MLIR_TUNING_CFG="path/to/config_file.cfg"`
`MIGRAPHX_MLIR_TUNE_LIMIT` Sets the maximum number of solutions available for MLIR tuning.	Takes an integer greater than 1.
`MIGRAPHX_MLIR_DUMP_TO_MXR` Sets the location to where the MXR files that the MLIR modules are written to are saved.	Takes the path to the directory where the files should be saved. For example: `MIGRAPHX_MLIR_DUMP_TO_MXR="/path/to/save_mxr_file/`
`MIGRAPHX_MLIR_DUMP` Sets the the location where the MLIR files that the MLIR modules are written to are saved.	Takes the path to the directory where the files should be saved. For example: `MIGRAPHX_MLIR_DUMP="/path/to/save_mlir_file/`

5.1.6 Testing

Environment variable	Values
`MIGRAPHX_TRACE_TEST_COMPIL` Sets the target to be traced, and turns on printing of the compile trace for verify tests on the given target. This flag cannot be used if `MIGRAPHX_TRACE_COMPILE` is used.	`cpu`: Turns on traces for the CPU target. `GPU`: Turns on traces for the GPU target. Default:
`MIGRAPHX_TRACE_TEST` When set, the reference and target programs are printed even if the verify tests pass.	`1`: The reference and target programs are printed when the verify tests pass. `0`: Returns to default behavior. Default: Reference and target programs aren't printed if the verify tests pass.
`MIGRAPHX_DUMP_TEST` When set, the model that is being verified using `test-verify` is output to an MXR file.	`1`: The model that is being verified is output to an MXR file. `0`: Returns to default behavior. Default: The model isn't output to file.
`MIGRAPHX_VERIFY_DUMP_DIFF` When set, writes out the output of the test results, as well as the reference, when they differ.	`1`: Test results are written out when they differ. `0`: Returns to default behavior. Default: The results and the reference aren't written out when they differ.

5.1.7 Advanced settings

Environment variable	Values
`MIGRAPHX_TRACE_CMD_EXECUTE` When set, commands run by the MIGraphX process will be printed.	`1`: Printing of commands is turned on. `0`: Returns to default behavior. Default: Commands aren't printed.
`MIGRAPHX_TRACE_HIPRTC` When set, the HIPRTC options and C++ file used will be printed.	`1`: HIPRTC options and C++ file will be printed. `0`: Returns to default behavior. Default: HIPRTC options and C++ file aren't printed.
`MIGRAPHX_DEBUG_SAVE_TEMP_DIR` When set, temporary directories won't be deleted.	`1`: Temporary directories aren't deleted. `0`: Returns to default behavior. Default: Temporary directories are deleted.
`MIGRAPHX_GPU_DEBUG` When set, the `-DMIGRAPHX_DEBUG` option is used when compiling GPU kernels. `-DMIGRAPHX_DEBUG` enables assertions and source location capture.	`1`: The `-DMIGRAPHX_DEBUG` option is used when compiling GPU kernels. Default: Compilation is run without `-DMIGRAPHX_DEBUG`.
`MIGRAPHX_GPU_DEBUG_SYM` When set, the `-g` option is used when compiling HIPRTC for debugging purposes.	`1`: The `-g` option is used when compiling HIPRTC. Default: Compilation is run without the `-g` option.
`MIGRAPHX_GPU_DUMP_SRC` The compiled HIPRTC source files is written out for further analysis.	`1`: HIPRTC source files are written out. `0`: Returns to default behavior. Default: HIPRTC source files aren't written out.
`MIGRAPHX_GPU_DUMP_ASM` When set, the hip-clang assembly output is written out for further analysis.	`1`: The hip-clang assembly output is written out. `0`: Returns to default behavior. Default: The hip-clang assembly output isn't written out.
`MIGRAPHX_GPU_HIP_FLAGS` When set, the hip-clang compiler appends these extra flags for compilation.	Takes a valid string, a valid hip compile option, e.g. "-Wno-error". Default: The compiler will not append any extra flags for compilation.

5.2 MIGraphX driver

The MIGraphX driver is a command-line tool that allows you to utilize many of the MIGraphX core functions without having to write a program. It can read, compile, run, and test the performance of a model with randomized data.

It is installed by default when you install MIGraphX. You can find it in `/opt/rocm/bin/migraphx-driver` or in `AMDMIGraphX/build/bin/migraphx-driver` after building the source code.

5.2.1 Commands

The table below summarizes the MIGraphX driver commands.

Table 1: commands

Command	Description
op	Prints all operators of MIGraphX when followed by the option `--list` or `-l`
params	Prints the input and output parameter shapes
run	Compiles, allocates parameters, evaluates, and prints input graph
read	Loads and prints input graph
compile	Compiles and prints input graph
verify	Runs reference and GPU implementations and checks outputs for consistency
perf	Compiles and runs input graph followed by printing the performance report

5.2.2 Options

The table below summarizes the various options to be used with the *MIGraphX driver commands*. To learn which options can be used with which commands, see the MIGraphX driver options.

Table 2: commands

Option	Description
–help | -h	Prints help section.
–test	Test MIGraphX with single layer GEMM model.
–onnx	Loads the file as an ONNX graph.
–tf	Loads the file as a tensorflow graph.
–migraphx	Loads the file as a migraphx graph.
–migraphx-json	Loads the file as a migraphx JSON graph.
–batch	Sets batch size for a static model. Sets the batch size at runtime for a dynamic batch model.
–nhwc	Treats tensorflow format as nhwc.
–nchw	Treats tensorflow format as nchw.
–skip-unknown-operators	Skips unknown operators when parsing and continues to parse.
–trim | -t	Trims instructions from the end.
–optimize | -O	Optimizes read
–graphviz | -g	Prints a graphviz representation
–brief	Makes the output brief
–cpp	Prints the program in .cpp format
–json	Prints the program in .json format
–text	Prints the program in .txt format
–binary	Prints the program in binary format
–netron	Prints the program in Netron viewable JSON format
–output | -o	Writes output in a file
–fill0	Fills parameter with 0s
–fill1	Fills parameter with 1s
–input-dim	Sets static dimensions of a parameter

continues on next page

Table 2 – continued from previous page

–dyn-input-dim	Sets dynamic dimensions of a parameter
–default-dyn-dim	Sets default dynamic dimension
–gpu	Compiles on the GPU
–cpu	Compiles on the CPU
–ref	Compiles on the reference implementation
–enable-offload-copy	Enables implicit offload copying
–disable-fast-math	Disables fast math optimization
–exhaustive-tune	Enables exhaustive search to find the fastest kernel
–fp16	Quantizes for fp16
–bf16	Quantizes for bf16
–int8	Quantizes for int8
–fp8	Quantize for `Float8E4M3FNUZ` type
–rms-tol	Sets tolerance for the RMS error (Default: 0.001)
–atol	Sets tolerance for elementwise absolute difference (Default: 0.001)
–rtol	Sets tolerance for elementwise relative difference (Default: 0.001)
–per-instruction | -i	Verifies each instruction
–reduce | -r	Reduces program and verifies
–iterations | -n	Sets the number of iterations to run for perf report
–list | -l	Lists all the MIGraphX operators

5.2.3 Usage

This section demonstrates the usage of MIGraphX driver tool with some commonly used options. Note that these examples use a simple MNIST ConvNet as the input graph for demonstration purposes as models of higher complexity generate considerably larger outputs in most cases.

Option: op

$ /opt/rocm/bin/migraphx-driver op –list

View Output

```
@literal
@param
@return
abs
acos
acosh
add
argmax
argmin
as_shape
asin
asinh
atan
atanh
batch_norm_inference
broadcast
capture
ceil
check_context::migraphx::gpu::context
clip
concat
```

(continues on next page)

(continued from previous page)

```
contiguous
convert
convolution
cos
cosh
deconvolution
div
dot
elu
equal
erf
exp
flatten
floor
gather
gpu::abs
gpu::acos
gpu::acosh
gpu::add
gpu::add_clip
gpu::add_gelu
gpu::add_gelu_new
gpu::add_relu
gpu::add_tanh
gpu::argmax
gpu::argmin
gpu::asin
gpu::asinh
gpu::atan
gpu::atanh
gpu::batch_norm_inference
gpu::ceil
gpu::clip
gpu::concat
gpu::contiguous
gpu::conv_bias
gpu::conv_bias_relu
gpu::convert
gpu::convolution
gpu::cos
gpu::cosh
gpu::deconv
gpu::div
gpu::elu
gpu::equal
gpu::erf
gpu::exp
gpu::floor
gpu::gather
gpu::gelu
gpu::gelu_new
gpu::gemm
```

(continues on next page)

(continued from previous page)

```
gpu::greater
gpu::layernorm
gpu::leaky_relu
gpu::less
gpu::log
gpu::logsoftmax
gpu::lrn
gpu::max
gpu::min
gpu::mul
gpu::mul_add
gpu::mul_add_relu
gpu::pad
gpu::pooling
gpu::pow
gpu::prelu
gpu::quant_convolution
gpu::quant_gemm
gpu::recip
gpu::record_event
gpu::reduce_max
gpu::reduce_mean
gpu::reduce_min
gpu::reduce_prod
gpu::reduce_sum
gpu::relu
gpu::rnn_var_sl_last_output
gpu::rnn_var_sl_shift_output
gpu::rnn_var_sl_shift_sequence
gpu::round
gpu::rsqrt
gpu::set_stream
gpu::sigmoid
gpu::sign
gpu::sin
gpu::sinh
gpu::softmax
gpu::sqdiff
gpu::sqrt
gpu::sub
gpu::tan
gpu::tanh
gpu::triadd
gpu::triadd_clip
gpu::triadd_relu
gpu::triadd_sigmoid
gpu::triadd_tanh
gpu::wait_event
greater
gru
hip::allocate
hip::copy
```

(continues on next page)

(continued from previous page)

```
hip::copy_from_gpu
hip::copy_to_gpu
hip::hip_allocate_memory
hip::hip_copy_literal
identity
im2col
leaky_relu
less
load
log
logsoftmax
lrn
lstm
max
min
mul
multibroadcast
neg
outline
pad
pooling
pow
prelu
quant_convolution
quant_dot
recip
reduce_max
reduce_mean
reduce_min
reduce_prod
reduce_sum
ref::batch_norm_inference
ref::convolution
ref::deconvolution
ref::dot
ref::elu
ref::im2col
ref::leaky_relu
ref::logsoftmax
ref::lrn
ref::op
ref::pad
ref::pooling_average
ref::pooling_max
ref::quant_convolution
ref::rnn_var_sl_last_output
ref::softmax
relu
reshape
rnn
rnn_last_cell_output
rnn_last_hs_output
```

(continues on next page)

(continued from previous page)

```
rnn_var_sl_last_output
rnn_var_sl_shift_output
rnn_var_sl_shift_sequence
round
rsqrt
scalar
sigmoid
sign
sin
sinh
slice
softmax
sqdiff
sqrt
squeeze
sub
tan
tanh
transpose
undefined
unknown:
unsqueeze
```

Option: params

$ /opt/rocm/bin/migraphx-driver params simple_graph.pb

View Output

```
Reading: simple_graph.pb
x: float_type, {1, 28, 28}, {784, 28, 1}
```

Option: run (ONNX file input)

$ /opt/rocm/bin/migraphx-driver run –onnx simple_graph.onnx

View Output

```
Compiling ...
Reading: simple_graph.onnx
@0 = check_context::migraphx::gpu::context -> float_type, {}, {}
@1 = hip::hip_allocate_memory[shape=float_type, {256}, {1},id=scratch] -> float_type,
↪{256}, {1}
@2 = hip::hip_copy_literal[id=@literal:1] -> float_type, {784, 128}, {128, 1}
x:0 = @param:x:0 -> float_type, {1, 28, 28}, {784, 28, 1}
@3 = reshape[dims={-1, 784}](x:0) -> float_type, {1, 784}, {784, 1}
@4 = load[offset=0,end=512](@1) -> float_type, {1, 128}, {128, 1}
@5 = gpu::gemm[alpha=1,beta=0](@3,@2,@4) -> float_type, {1, 128}, {128, 1}
@6 = hip::hip_copy_literal[id=@literal:0] -> float_type, {128}, {1}
@7 = hip::hip_copy_literal[id=@literal:2] -> float_type, {10}, {1}
@8 = hip::hip_copy_literal[id=@literal:3] -> float_type, {128, 10}, {10, 1}
@9 = multibroadcast[output_lens={1, 128}](@6) -> float_type, {1, 128}, {0, 1}
@10 = load[offset=512,end=1024](@1) -> float_type, {1, 128}, {128, 1}
```

(continues on next page)

(continued from previous page)

```
@11 = gpu::add_relu(@5,@9,@10) -> float_type, {1, 128}, {128, 1}
@12 = load[offset=0,end=40](@1) -> float_type, {1, 10}, {10, 1}
@13 = gpu::gemm[alpha=1,beta=0](@11,@8,@12) -> float_type, {1, 10}, {10, 1}
@14 = multibroadcast[output_lens={1, 10}](@7) -> float_type, {1, 10}, {0, 1}
@15 = load[offset=40,end=80](@1) -> float_type, {1, 10}, {10, 1}
@16 = gpu::add(@13,@14,@15) -> float_type, {1, 10}, {10, 1}
#output_0 = @param:#output_0 -> float_type, {1, 10}, {10, 1}
@17 = gpu::softmax[axis=1](@16,#output_0) -> float_type, {1, 10}, {10, 1}
@18 = @return(@17)

Allocating params ...
@0 = check_context::migraphx::gpu::context -> float_type, {}, {}
@1 = hip::hip_allocate_memory[shape=float_type, {256}, {1},id=scratch] -> float_type,
→{256}, {1}
@2 = hip::hip_copy_literal[id=@literal:1] -> float_type, {784, 128}, {128, 1}
x:0 = @param:x:0 -> float_type, {1, 28, 28}, {784, 28, 1}
@3 = reshape[dims={-1, 784}](x:0) -> float_type, {1, 784}, {784, 1}
@4 = load[offset=0,end=512](@1) -> float_type, {1, 128}, {128, 1}
@5 = gpu::gemm[alpha=1,beta=0](@3,@2,@4) -> float_type, {1, 128}, {128, 1}
@6 = hip::hip_copy_literal[id=@literal:0] -> float_type, {128}, {1}
@7 = hip::hip_copy_literal[id=@literal:2] -> float_type, {10}, {1}
@8 = hip::hip_copy_literal[id=@literal:3] -> float_type, {128, 10}, {10, 1}
@9 = multibroadcast[output_lens={1, 128}](@6) -> float_type, {1, 128}, {0, 1}
@10 = load[offset=512,end=1024](@1) -> float_type, {1, 128}, {128, 1}
@11 = gpu::add_relu(@5,@9,@10) -> float_type, {1, 128}, {128, 1}
@12 = load[offset=0,end=40](@1) -> float_type, {1, 10}, {10, 1}
@13 = gpu::gemm[alpha=1,beta=0](@11,@8,@12) -> float_type, {1, 10}, {10, 1}
@14 = multibroadcast[output_lens={1, 10}](@7) -> float_type, {1, 10}, {0, 1}
@15 = load[offset=40,end=80](@1) -> float_type, {1, 10}, {10, 1}
@16 = gpu::add(@13,@14,@15) -> float_type, {1, 10}, {10, 1}
#output_0 = @param:#output_0 -> float_type, {1, 10}, {10, 1}
@17 = gpu::softmax[axis=1](@16,#output_0) -> float_type, {1, 10}, {10, 1}
@18 = @return(@17)
```

Option: read

$ /opt/rocm/bin/migraphx-driver read simple_graph.pb

View Output

```
Reading: simple_graph.pb
@0 = @literal{0.0136018, -0.0839988, 0.0375392, 0.0613085, -0.125795, 0.176185, 0.
→0761055, 0.0093384, -0.110057, -0.170587} -> float_type, {10}, {1}
@1 = @literal{ ... } -> float_type, {128, 10}, {10, 1}
@2 = @literal{ ... } -> float_type, {128}, {1}
@3 = @literal{ ... } -> float_type, {784, 128}, {128, 1}
@4 = @literal{-1, 784} -> int32_type, {2}, {1}
x = @param:x -> float_type, {1, 28, 28}, {784, 28, 1}
@5 = reshape[dims={-1, 784}](x) -> float_type, {1, 784}, {784, 1}
@6 = identity(@3) -> float_type, {784, 128}, {128, 1}
@7 = dot[alpha=1,beta=1](@5,@6) -> float_type, {1, 128}, {128, 1}
@8 = identity(@2) -> float_type, {128}, {1}
```

(continues on next page)

(continued from previous page)

```
@9 = broadcast[axis=1,dims={1, 128}](@8) -> float_type, {1, 128}, {0, 1}
@10 = add(@7,@9) -> float_type, {1, 128}, {128, 1}
@11 = relu(@10) -> float_type, {1, 128}, {128, 1}
@12 = identity(@1) -> float_type, {128, 10}, {10, 1}
@13 = dot[alpha=1,beta=1](@11,@12) -> float_type, {1, 10}, {10, 1}
@14 = identity(@0) -> float_type, {10}, {1}
@15 = broadcast[axis=1,dims={1, 10}](@14) -> float_type, {1, 10}, {0, 1}
@16 = add(@13,@15) -> float_type, {1, 10}, {10, 1}
@17 = softmax[axis=1](@16) -> float_type, {1, 10}, {10, 1}
@18 = identity(@17) -> float_type, {1, 10}, {10, 1}
```

Option: compile (on GPU, quantized for fp16)

$ /opt/rocm/bin/migraphx-driver compile –gpu –fp16 simple_graph.pb

View Output

```
Compiling ...
Reading: simple_graph.pb
@0 = check_context::migraphx::gpu::context -> float_type, {}, {}
@1 = hip::hip_allocate_memory[shape=float_type, {456}, {1},id=scratch] -> float_type,
↪{456}, {1}
@2 = hip::hip_copy_literal[id=@literal:0] -> half_type, {784, 128}, {128, 1}
@3 = load[offset=256,end=1824](@1) -> half_type, {1, 28, 28}, {784, 28, 1}
x = @param:x -> float_type, {1, 28, 28}, {784, 28, 1}
@4 = gpu::convert[target_type=1](x,@3) -> half_type, {1, 28, 28}, {784, 28, 1}
@5 = reshape[dims={-1, 784}](@4) -> half_type, {1, 784}, {784, 1}
@6 = load[offset=0,end=256](@1) -> half_type, {1, 128}, {128, 1}
@7 = gpu::gemm[alpha=1,beta=0](@5,@2,@6) -> half_type, {1, 128}, {128, 1}
@8 = hip::hip_copy_literal[id=@literal:2] -> half_type, {128, 10}, {10, 1}
@9 = hip::hip_copy_literal[id=@literal:1] -> half_type, {128}, {1}
@10 = hip::hip_copy_literal[id=@literal:3] -> half_type, {10}, {1}
@11 = load[offset=256,end=512](@1) -> half_type, {1, 128}, {128, 1}
@12 = broadcast[axis=1,dims={1, 128}](@9) -> half_type, {1, 128}, {0, 1}
@13 = gpu::add_relu(@7,@12,@11) -> half_type, {1, 128}, {128, 1}
@14 = load[offset=0,end=20](@1) -> half_type, {1, 10}, {10, 1}
@15 = gpu::gemm[alpha=1,beta=0](@13,@8,@14) -> half_type, {1, 10}, {10, 1}
@16 = broadcast[axis=1,dims={1, 10}](@10) -> half_type, {1, 10}, {0, 1}
@17 = load[offset=20,end=40](@1) -> half_type, {1, 10}, {10, 1}
@18 = gpu::add(@15,@16,@17) -> half_type, {1, 10}, {10, 1}
@19 = load[offset=0,end=20](@1) -> half_type, {1, 10}, {10, 1}
@20 = gpu::softmax[axis=1](@18,@19) -> half_type, {1, 10}, {10, 1}
output = @param:output -> float_type, {1, 10}, {10, 1}
@21 = gpu::convert[target_type=2](@20,output) -> float_type, {1, 10}, {10, 1}
```

Option: verify

$ /opt/rocm/bin/migraphx-driver verify simple_graph.pb

View Output

```
Reading: simple_graph.pb
@0 = @literal{0.0136018, -0.0839988, 0.0375392, 0.0613085, -0.125795, 0.176185, 0.
```

(continues on next page)

(continued from previous page)

```
↪0761055, 0.0093384, -0.110057, -0.170587} -> float_type, {10}, {1}
@1 = @literal{ ... } -> float_type, {128, 10}, {10, 1}
@2 = @literal{ ... } -> float_type, {128}, {1}
@3 = @literal{ ... } -> float_type, {784, 128}, {128, 1}
@4 = @literal{-1, 784} -> int32_type, {2}, {1}
x = @param:x -> float_type, {1, 28, 28}, {784, 28, 1}
@5 = reshape[dims={-1, 784}](x) -> float_type, {1, 784}, {784, 1}
@6 = identity(@3) -> float_type, {784, 128}, {128, 1}
@7 = dot[alpha=1,beta=1](@5,@6) -> float_type, {1, 128}, {128, 1}
@8 = identity(@2) -> float_type, {128}, {1}
@9 = broadcast[axis=1,dims={1, 128}](@8) -> float_type, {1, 128}, {0, 1}
@10 = add(@7,@9) -> float_type, {1, 128}, {128, 1}
@11 = relu(@10) -> float_type, {1, 128}, {128, 1}
@12 = identity(@1) -> float_type, {128, 10}, {10, 1}
@13 = dot[alpha=1,beta=1](@11,@12) -> float_type, {1, 10}, {10, 1}
@14 = identity(@0) -> float_type, {10}, {1}
@15 = broadcast[axis=1,dims={1, 10}](@14) -> float_type, {1, 10}, {0, 1}
@16 = add(@13,@15) -> float_type, {1, 10}, {10, 1}
@17 = softmax[axis=1](@16) -> float_type, {1, 10}, {10, 1}
@18 = identity(@17) -> float_type, {1, 10}, {10, 1}

@0 = @literal{0.0136018, -0.0839988, 0.0375392, 0.0613085, -0.125795, 0.176185, 0.
↪0761055, 0.0093384, -0.110057, -0.170587} -> float_type, {10}, {1}
@1 = @literal{ ... } -> float_type, {128, 10}, {10, 1}
@2 = @literal{ ... } -> float_type, {128}, {1}
@3 = @literal{ ... } -> float_type, {784, 128}, {128, 1}
@4 = @literal{-1, 784} -> int32_type, {2}, {1}
x = @param:x -> float_type, {1, 28, 28}, {784, 28, 1}
@5 = reshape[dims={-1, 784}](x) -> float_type, {1, 784}, {784, 1}
@6 = identity(@3) -> float_type, {784, 128}, {128, 1}
@7 = dot[alpha=1,beta=1](@5,@6) -> float_type, {1, 128}, {128, 1}
@8 = identity(@2) -> float_type, {128}, {1}
@9 = broadcast[axis=1,dims={1, 128}](@8) -> float_type, {1, 128}, {0, 1}
@10 = add(@7,@9) -> float_type, {1, 128}, {128, 1}
@11 = relu(@10) -> float_type, {1, 128}, {128, 1}
@12 = identity(@1) -> float_type, {128, 10}, {10, 1}
@13 = dot[alpha=1,beta=1](@11,@12) -> float_type, {1, 10}, {10, 1}
@14 = identity(@0) -> float_type, {10}, {1}
@15 = broadcast[axis=1,dims={1, 10}](@14) -> float_type, {1, 10}, {0, 1}
@16 = add(@13,@15) -> float_type, {1, 10}, {10, 1}
@17 = softmax[axis=1](@16) -> float_type, {1, 10}, {10, 1}
@18 = identity(@17) -> float_type, {1, 10}, {10, 1}

@0 = @literal{0.0136018, -0.0839988, 0.0375392, 0.0613085, -0.125795, 0.176185, 0.
↪0761055, 0.0093384, -0.110057, -0.170587} -> float_type, {10}, {1}
@1 = @literal{ ... } -> float_type, {128, 10}, {10, 1}
@2 = @literal{ ... } -> float_type, {128}, {1}
@3 = @literal{ ... } -> float_type, {784, 128}, {128, 1}
x = @param:x -> float_type, {1, 28, 28}, {784, 28, 1}
@4 = ref::reshape[dims={-1, 784}](x) -> float_type, {1, 784}, {784, 1}
@5 = ref::identity(@3) -> float_type, {784, 128}, {128, 1}
@6 = ref::dot[alpha=1,beta=1](@4,@5) -> float_type, {1, 128}, {128, 1}
```

(continues on next page)

(continued from previous page)

```
@7 = ref::identity(@2) -> float_type, {128}, {1}
@8 = ref::broadcast[axis=1,dims={1, 128}](@7) -> float_type, {1, 128}, {0, 1}
@9 = ref::contiguous(@8) -> float_type, {1, 128}, {128, 1}
@10 = ref::add(@6,@9) -> float_type, {1, 128}, {128, 1}
@11 = ref::relu(@10) -> float_type, {1, 128}, {128, 1}
@12 = ref::identity(@1) -> float_type, {128, 10}, {10, 1}
@13 = ref::dot[alpha=1,beta=1](@11,@12) -> float_type, {1, 10}, {10, 1}
@14 = ref::identity(@0) -> float_type, {10}, {1}
@15 = ref::broadcast[axis=1,dims={1, 10}](@14) -> float_type, {1, 10}, {0, 1}
@16 = ref::contiguous(@15) -> float_type, {1, 10}, {10, 1}
@17 = ref::add(@13,@16) -> float_type, {1, 10}, {10, 1}
@18 = ref::softmax[axis=1](@17) -> float_type, {1, 10}, {10, 1}
@19 = ref::identity(@18) -> float_type, {1, 10}, {10, 1}

@0 = check_context::migraphx::gpu::context -> float_type, {}, {}
@1 = hip::hip_allocate_memory[shape=float_type, {256}, {1},id=scratch] -> float_type,
↪{256}, {1}
@2 = hip::hip_copy_literal[id=@literal:3] -> float_type, {784, 128}, {128, 1}
x = @param:x -> float_type, {1, 28, 28}, {784, 28, 1}
@3 = load[offset=0,end=512](@1) -> float_type, {1, 128}, {128, 1}
@4 = reshape[dims={-1, 784}](x) -> float_type, {1, 784}, {784, 1}
@5 = gpu::gemm[alpha=1,beta=0](@4,@2,@3) -> float_type, {1, 128}, {128, 1}
@6 = hip::hip_copy_literal[id=@literal:1] -> float_type, {128, 10}, {10, 1}
@7 = hip::hip_copy_literal[id=@literal:2] -> float_type, {128}, {1}
@8 = hip::hip_copy_literal[id=@literal:0] -> float_type, {10}, {1}
@9 = load[offset=512,end=1024](@1) -> float_type, {1, 128}, {128, 1}
@10 = broadcast[axis=1,dims={1, 128}](@7) -> float_type, {1, 128}, {0, 1}
@11 = gpu::add_relu(@5,@10,@9) -> float_type, {1, 128}, {128, 1}
@12 = load[offset=40,end=80](@1) -> float_type, {1, 10}, {10, 1}
@13 = gpu::gemm[alpha=1,beta=0](@11,@6,@12) -> float_type, {1, 10}, {10, 1}
@14 = load[offset=0,end=40](@1) -> float_type, {1, 10}, {10, 1}
@15 = broadcast[axis=1,dims={1, 10}](@8) -> float_type, {1, 10}, {0, 1}
@16 = gpu::add(@13,@15,@14) -> float_type, {1, 10}, {10, 1}
output = @param:output -> float_type, {1, 10}, {10, 1}
@17 = gpu::softmax[axis=1](@16,output) -> float_type, {1, 10}, {10, 1}
```

Option: perf

$ /opt/rocm/bin/migraphx-driver perf simple_graph.pb

View Output

```
Compiling ...
Reading: simple_graph.pb
@0 = check_context::migraphx::gpu::context -> float_type, {}, {}
@1 = hip::hip_allocate_memory[shape=float_type, {256}, {1},id=scratch] -> float_type,
↪{256}, {1}
@2 = hip::hip_copy_literal[id=@literal:3] -> float_type, {784, 128}, {128, 1}
@3 = load[offset=0,end=512](@1) -> float_type, {1, 128}, {128, 1}
x = @param:x -> float_type, {1, 28, 28}, {784, 28, 1}
@4 = reshape[dims={-1, 784}](x) -> float_type, {1, 784}, {784, 1}
@5 = gpu::gemm[alpha=1,beta=0](@4,@2,@3) -> float_type, {1, 128}, {128, 1}
```

(continues on next page)

(continued from previous page)

```
@6 = hip::hip_copy_literal[id=@literal:1] -> float_type, {128, 10}, {10, 1}
@7 = hip::hip_copy_literal[id=@literal:0] -> float_type, {10}, {1}
@8 = hip::hip_copy_literal[id=@literal:2] -> float_type, {128}, {1}
@9 = broadcast[axis=1,dims={1, 128}](@8) -> float_type, {1, 128}, {0, 1}
@10 = load[offset=512,end=1024](@1) -> float_type, {1, 128}, {128, 1}
@11 = gpu::add_relu(@5,@9,@10) -> float_type, {1, 128}, {128, 1}
@12 = load[offset=0,end=40](@1) -> float_type, {1, 10}, {10, 1}
@13 = gpu::gemm[alpha=1,beta=0](@11,@6,@12) -> float_type, {1, 10}, {10, 1}
@14 = broadcast[axis=1,dims={1, 10}](@7) -> float_type, {1, 10}, {0, 1}
@15 = load[offset=40,end=80](@1) -> float_type, {1, 10}, {10, 1}
@16 = gpu::add(@13,@14,@15) -> float_type, {1, 10}, {10, 1}
output = @param:output -> float_type, {1, 10}, {10, 1}
@17 = gpu::softmax[axis=1](@16,output) -> float_type, {1, 10}, {10, 1}

Allocating params ...
Running performance report ...
@0 = check_context::migraphx::gpu::context -> float_type, {}, {}: 0.00057782ms, 1%
@1 = hip::hip_allocate_memory[shape=float_type, {256}, {1},id=scratch] -> float_type,
↪{256}, {1}: 0.000295ms, 1%
@2 = hip::hip_copy_literal[id=@literal:3] -> float_type, {784, 128}, {128, 1}: 0.
↪00027942ms, 1%
@3 = load[offset=0,end=512](@1) -> float_type, {1, 128}, {128, 1}: 0.000232ms, 1%
x = @param:x -> float_type, {1, 28, 28}, {784, 28, 1}: 0.0003206ms, 1%
@4 = reshape[dims={-1, 784}](x) -> float_type, {1, 784}, {784, 1}: 0.00033842ms, 1%
@5 = gpu::gemm[alpha=1,beta=0](@4,@2,@3) -> float_type, {1, 128}, {128, 1}: 0.212592ms,␣
↪52%
@6 = hip::hip_copy_literal[id=@literal:1] -> float_type, {128, 10}, {10, 1}: 0.
↪00085822ms, 1%
@7 = hip::hip_copy_literal[id=@literal:0] -> float_type, {10}, {1}: 0.000382ms, 1%
@8 = hip::hip_copy_literal[id=@literal:2] -> float_type, {128}, {1}: 0.0003486ms, 1%
@9 = broadcast[axis=1,dims={1, 128}](@8) -> float_type, {1, 128}, {0, 1}: 0.000299ms, 1%
@10 = load[offset=512,end=1024](@1) -> float_type, {1, 128}, {128, 1}: 0.000234ms, 1%
@11 = gpu::add_relu(@5,@9,@10) -> float_type, {1, 128}, {128, 1}: 0.0416597ms, 11%
@12 = load[offset=0,end=40](@1) -> float_type, {1, 10}, {10, 1}: 0.0007548ms, 1%
@13 = gpu::gemm[alpha=1,beta=0](@11,@6,@12) -> float_type, {1, 10}, {10, 1}: 0.0733071ms,
↪ 18%
@14 = broadcast[axis=1,dims={1, 10}](@7) -> float_type, {1, 10}, {0, 1}: 0.00088142ms, 1%
@15 = load[offset=40,end=80](@1) -> float_type, {1, 10}, {10, 1}: 0.000408ms, 1%
@16 = gpu::add(@13,@14,@15) -> float_type, {1, 10}, {10, 1}: 0.0410144ms, 10%
output = @param:output -> float_type, {1, 10}, {10, 1}: 0.0010222ms, 1%
@17 = gpu::softmax[axis=1](@16,output) -> float_type, {1, 10}, {10, 1}: 0.0385636ms, 10%

Summary:
gpu::gemm: 0.285899ms, 69%
gpu::add_relu: 0.0416597ms, 11%
gpu::add: 0.0410144ms, 10%
gpu::softmax: 0.0385636ms, 10%
hip::hip_copy_literal: 0.00186824ms, 1%
load: 0.0016288ms, 1%
@param: 0.0013428ms, 1%
broadcast: 0.00118042ms, 1%
check_context::migraphx::gpu::context: 0.00057782ms, 1%
```

(continues on next page)

(continued from previous page)

```
reshape: 0.00033842ms, 1%
hip::hip_allocate_memory: 0.000295ms, 1%

Rate: 2866.1/sec
Total time: 0.348906ms
Total instructions time: 0.414369ms
Overhead time: 0.00348144ms, -0.0654627ms
Overhead: 1%, -19%
```

5.3 Developing for MIGraphX

This document is intended for anyone who wants to contribute to MIGraphX. This document covers some basic operations that can be used to develop for MIGraphX. The complete source code for the example shown here can be found at ref_dev_examples.cpp on the MIGraphX repository.

More examples can be found on the MIGraphX GitHub repository.

5.3.1 Adding two literals

A program is a collection of modules, which are collections of instructions to be executed when calling *eval*. Each instruction has an associated *operation* which represents the computation to be performed by the instruction.

We start with a snippet of the simple `add_two_literals()` function:

```
// create the program and get a pointer to the main module
migraphx::program p;
auto* mm = p.get_main_module();

// add two literals to the program
auto one = mm->add_literal(1);
auto two = mm->add_literal(2);

// make the add operation between the two literals and add it to the program
mm->add_instruction(migraphx::make_op("add"), one, two);

// compile the program on the reference device
p.compile(migraphx::ref::target{});

// evaluate the program and retreive the result
auto result = p.eval({}).back();
std::cout << "add_two_literals: 1 + 2 = " << result << "\n";
```

In the above function, a simple *program* object is created along with a pointer to the main module of it. The program is a collection of modules which starts execution from the main module, so instructions are added to the modules rather than the program object directly. The *add_literal* function is used to add an instruction that stores the literal number 1 while returning an *instruction_ref*. The returned *instruction_ref* can be used in another instruction as an input. The same *add_literal* function is used to add the literal 2 to the program. After the literals are created, the instruction is created to add the numbers. This is done by using the *add_instruction* function with the add *operation* created by `make_op` and the previously created literals passed as the arguments for the instruction. You can run this *program* by compiling it for the reference target (CPU) and then running it with *eval*. This prints the result on the console.

To compile the program for the GPU, move the file to `test/gpu/` directory and include the given target:

```
#include <migraphx/gpu/target.hpp>
```

5.3.2 Adding Parameters

While the add_two_literals() function above demonstrates add operation on constant values 1 and 2, the following program demonstrates how to pass a parameter (x) to a module using add_parameter() function .

> migraphx::program p; auto* mm = p.get_main_module(); migraphx::shape s{migraphx::shape::int32_type, {1}};
>
> // add parameter "x" with the shape s auto x = mm->add_parameter("x", s); auto two = mm->add_literal(2);
>
> // add the "add" instruction between the "x" parameter and "two" to the module mm->add_instruction(migraphx::make_op("add"), x, two); p.compile(migraphx::ref::target{});

In the code snippet above, an add operation is performed on a parameter of type int32 and literal 2 followed by compilation for the CPU. To run the program, pass the parameter as a parameter_map while calling *eval*. To map the parameter x to an *argument* object with an int data type, a parameter_map is created as shown below:

```
// create a parameter_map object for passing a value to the "x" parameter
std::vector<int> data = {4};
migraphx::parameter_map params;
params["x"] = migraphx::argument(s, data.data());

auto result = p.eval(params).back();
std::cout << "add_parameters: 4 + 2 = " << result << "\n";
EXPECT(result.at<int>() == 6);
```

5.3.3 Handling Tensor Data

The above two examples demonstrate scalar operations. To describe multi-dimensional tensors, use the *shape* class to compute a simple convolution as shown below:

```
migraphx::program p;
auto* mm = p.get_main_module();

// create shape objects for the input tensor and weights
migraphx::shape input_shape{migraphx::shape::float_type, {2, 3, 4, 4}};
migraphx::shape weights_shape{migraphx::shape::float_type, {3, 3, 3, 3}};

// create the parameters and add the "convolution" operation to the module
auto input   = mm->add_parameter("X", input_shape);
auto weights = mm->add_parameter("W", weights_shape);
mm->add_instruction(migraphx::make_op("convolution", {{"padding", {1, 1}}, {"stride", {2,
↪ 2}}}), input, weights);
```

Most programs take data from allocated buffers that are usually on the GPU. To pass the buffer data as an argument, create *argument* objects directly from the pointers to the buffers:

```
// Compile the program
p.compile(migraphx::ref::target{});

// Allocated buffers by the user
std::vector<float> a = ...;
```

(continues on next page)

(continued from previous page)

```
std::vector<float> c = ...;

// Solution vector
std::vector<float> sol = ...;

// Create the arguments in a parameter_map
migraphx::parameter_map params;
params["X"] = migraphx::argument(input_shape, a.data());
params["W"] = migraphx::argument(weights_shape, c.data());

// Evaluate and confirm the result
auto result = p.eval(params).back();
std::vector<float> results_vector(64);
result.visit([&](auto output) { results_vector.assign(output.begin(), output.end()); });

EXPECT(migraphx::verify::verify_rms_range(results_vector, sol));
```

An `argument` can handle memory buffers from either the GPU or the CPU. When running the `program`, buffers are allocated on the corresponding target by default. By default, the buffers are allocated on the CPU when compiling for CPU and on the GPU when compiling for GPU. To locate the buffers on the CPU even when compiling for GPU, set the option `offload_copy=true`.

5.3.4 Importing From ONNX

To make it convenient to use neural networks directly from other frameworks, MIGraphX ONNX parser allows you to build a `program` directly from an ONNX file. For usage, refer to the `parse_onnx()` function below:

```
program p = migraphx::parse_onnx("model.onnx");
p.compile(migraphx::gpu::target{});
```

5.3.5 Build this example

Build the ref_dev_examples.cpp example with this command:

> make -j$(nproc) test_ref_dev_examples

This creates the `test_ref_dev_examples` under `bin/` in the build directory.

To verify the build, use:

> make -j$(nproc) check

5.4 Data types

5.4.1 shape

struct **shape**

Public Types

enum **type_t**

Values:

enumerator **bool_type**

enumerator **half_type**

enumerator **float_type**

enumerator **double_type**

enumerator **uint8_type**

enumerator **int8_type**

enumerator **uint16_type**

enumerator **int16_type**

enumerator **int32_type**

enumerator **int64_type**

enumerator **uint32_type**

enumerator **uint64_type**

enumerator **fp8e4m3fnuz_type**

enumerator **fp8e4m3fn_type**

enumerator **fp8e5m2_type**

enumerator **bf16_type**

enumerator **fp8e5m2fnuz_type**

enumerator **tuple_type**

enumerator **fp4x2_type**

Public Functions

shape()

shape(*type_t* t)

shape(*type_t* t, std::vector<std::size_t> l)

shape(*type_t* t, std::vector<std::size_t> l, std::vector<std::size_t> s)

shape(*type_t* t, std::initializer_list<std::size_t> d)

shape(*type_t* t, std::vector<*dynamic_dimension*> dims)

shape(*type_t* t, std::vector<std::size_t> mins, std::vector<std::size_t> maxes, std::vector<std::set<std::size_t>> optimals_list)

template<class **Range**>
inline **shape**(*type_t* t, const *Range* &l)

template<class **Range1**, class **Range2**>
inline **shape**(*type_t* t, const *Range1* &l, const *Range2* &s)

explicit **shape**(const std::vector<*shape*> &subs)

type_t **type**() const

const std::vector<std::size_t> &**lens**() const

const std::vector<std::size_t> &**strides**() const

std::size_t **ndim**() const

The number of dimensions in the shape, either static or dynamic. Same as the number of indices required to get a data value.

std::size_t **elements**() const

Return the number of elements in the tensor.

std::size_t **bytes**() const

Return the number of total bytes used for storage of the tensor data; includes subshapes. For dynamic shape, returns the maximum number of bytes presuming a packed shape.

std::size_t **type_size**() const

Return the size of the type of the main shape. Returns 0 if there are subshapes.

const std::vector<*dynamic_dimension*> &**dyn_dims**() const

std::vector<std::size_t> **min_lens**() const

Minimum lengths for dynamic shape. *lens()* for static shape.

std::vector<std::size_t> **max_lens**() const

Maximum lengths for dynamic shape. *lens()* for static shape.

std::vector<std::set<std::size_t>> **opt_lens**() const

Optimum lengths for dynamic shape. Empty for static shape.

std::size_t **index**(std::initializer_list<std::size_t> l) const

Map multiple indices to space index.

std::size_t **index**(const std::vector<std::size_t> &l) const

Map multiple indices to space index.

template<class **Iterator**>
inline std::size_t **index**(*Iterator* start, *Iterator* last) const

Map multiple indices from a range of iterator to a space index.

std::size_t **index**(std::size_t i) const

Map element index to space index.

std::vector<std::size_t> **multi**(std::size_t idx) const

Map element index to multi-dimensional index.

void **multi_copy**(std::size_t idx, std::size_t *start, const std::size_t *end) const

Map element index to multi-dimensional index and put them them into location provided by pointers

bool **multi_within_bounds**(std::vector<std::size_t> multi) const

Check if a multi-dimensional index is within bounds for the shape.

template<class **Iterator**>
inline std::size_t **single**(*Iterator* start, *Iterator* last) const

Convert multi-dimensional index into a single element index.

std::size_t **single**(const std::vector<std::size_t> &idx) const

Convert multi-dimensional index into a single element index.

bool **packed**() const

Returns true if the shape is packed (number of elements and buffer size the same) with no padding

bool **transposed**() const

Returns true if the shape has been transposed. That is the strides are not in descending order

bool **broadcasted**() const

Returns true if the shape is broadcasting a dimension. That is, one of the strides are zero.

bool **standard**() const

Returns true if the shape is in its standard format. That is, the shape is both packed and not transposed.

bool **scalar**() const

Returns true if all strides are equal to 0 (scalar tensor)

bool **dynamic**() const

Return true if the shape is dynamic.

bool **any_of_dynamic**() const

Return true if this shape or any of the sub_shapes are dynamic.

bool **computable**() const

If type is computable (can do math ops like add or divide) and has a visitor function.

shape **normalize_standard**() const

shape **as_standard**() const

shape **with_lens**(*type_t* t, const std::vector<std::size_t> &l) const

shape **with_lens**(const std::vector<std::size_t> &l) const

shape **with_type**(*type_t* t) const

shape **to_dynamic**() const

shape **to_static**(std::size_t x) const

template<class ...**Visitors**>
inline void **visit_type**(*Visitors*... vs) const

std::string **type_string**() const

const std::vector<*shape*> &**sub_shapes**() const

std::size_t **tuple_size**() const

std::size_t **element_space**() const

Returns the number of elements in the data buffer. For a dynamic shape, returns the maximum number of elements of the data buffer and assumes it is packed. Will clip to the maximum of size_t if overflows for dynamic shapes.

Public Static Functions

static std::string **to_sizes_string**(const std::vector<*shape*> &shapes)

static const std::vector<*type_t*> &**types**()

static std::string **name**(*type_t* t)

static std::string **cpp_type**(*type_t* t)

static bool **is_integral**(*type_t* t)

static bool **is_compatible**(const *shape* &actual, const *shape* &expected)

static bool **is_unsigned**(*type_t* t)

static bool **is_computable**(*type_t* t)

static *shape* **from_permutation**(*type_t* t, const std::vector<std::size_t> &l, const std::vector<int64_t> &perm)

Creates an output shape with dimensions `l` and strides computed to fulfill the given permutation.

`t` = shape type `l` = output dimensions `perm` = order dimensions from slowest dimension to fastest dimension

Example: `t` = float_type, `l` = [2, 3, 4], `perm` = [1, 2, 0] axis=1 to slowest dimension, axis=2 to second slowest, axis=0 to fastest returns shape{type = float, lens = [2, 3, 4], strides = [1, 8 ,2]}

template<class **Visitor**, class **TupleVisitor**>
static inline void **visit**(*type_t* t, *Visitor* v, *TupleVisitor* tv)

template<class **Visitor**>
static inline void **visit**(*type_t* t, *Visitor* v)

template<class **Visitor**>
static inline void **visit_types**(*Visitor* v)

static *type_t* **parse_type**(const std::string &s)

Friends

friend bool **operator==**(const *shape* &x, const *shape* &y)

friend bool **operator!=**(const *shape* &x, const *shape* &y)

friend std::ostream &**operator<<**(std::ostream &os, const *shape* &x)

template<class **T**>

struct **as**

Public Types

using **type** = std::conditional_t<std::is_same<*T*, bool>{}, int8_t, *T*>

Public Functions

inline *type* **max**() const

inline *type* **min**() const

inline *type* **nan**() const

inline *type* **epsilon**() const

template<class **U**>
inline *type* **operator()**(*U* u) const

template<class **U**>
inline *type* ***operator()**(*U* *u) const

template<class **U**>
inline const *type* ***operator()**(const *U* *u) const

inline *type* **operator()**() const

inline std::size_t **size**(std::size_t n = 1) const

inline bool **is_integral**() const

inline bool **is_signed**() const

inline bool **is_unsigned**() const

template<class **U**>
inline *type* ***from**(*U* *buffer, std::size_t n = 0) const

template<class **U**>
inline const *type* ***from**(const *U* *buffer, std::size_t n = 0) const

inline *type_t* **type_enum**() const

struct **dynamic_dimension**

Public Functions

bool **is_fixed**() const

bool **has_optimal**() const

inline std::optional<*dynamic_dimension*> **intersection**(const *dynamic_dimension* &other) const

Return a *dynamic_dimension* with the intersection of two *dynamic_dimension* ranges if possible.

dynamic_dimension &**operator+=**(const std::size_t &x)

dynamic_dimension &**operator-=**(const std::size_t &x)

dynamic_dimension &**operator*=**(const std::size_t &x)

Public Members

std::size_t **min** = 0

std::size_t **max** = 0

std::set<std::size_t> **optimals** = {}

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

Friends

friend bool **operator==**(const *dynamic_dimension* &x, const *dynamic_dimension* &y)

friend bool **operator!=**(const *dynamic_dimension* &x, const *dynamic_dimension* &y)

friend std::ostream &**operator<<**(std::ostream &os, const *dynamic_dimension* &x)

friend bool **operator==**(const *dynamic_dimension* &x, const std::size_t &y)

friend bool **operator==**(const std::size_t &x, const *dynamic_dimension* &y)

friend bool **operator!=**(const *dynamic_dimension* &x, const std::size_t &y)

friend bool **operator!=**(const std::size_t &x, const *dynamic_dimension* &y)

friend *dynamic_dimension* **operator+**(const *dynamic_dimension* &x, const std::size_t &y)

friend *dynamic_dimension* **operator+**(const std::size_t &x, const *dynamic_dimension* &y)

friend *dynamic_dimension* **operator-**(const *dynamic_dimension* &x, const std::size_t &y)

friend *dynamic_dimension* **operator***(const *dynamic_dimension* &x, const std::size_t &y)

friend *dynamic_dimension* **operator***(const std::size_t &x, const *dynamic_dimension* &y)

template<class **T**, class = void>

struct **get_type**

Subclassed by *migraphx::internal::shape::get_type< const T >*

template<class **T**>

struct **get_type**<bf16, *T*> : public std::integral_constant<*type_t*, bf16_type>

template<class **T**>

struct **get_type**<bool, *T*> : public std::integral_constant<*type_t*, bool_type>

template<class **T**>

struct **get_type**<const *T*> : public migraphx::internal::*shape*::*get_type*<*T*>

template<class **T**>

struct **get_type**<double, *T*> : public std::integral_constant<*type_t*, double_type>

template<class **T**>

struct **get_type**<float, *T*> : public std::integral_constant<*type_t*, float_type>

template<class **T**>

struct **get_type**<half, *T*> : public std::integral_constant<*type_t*, half_type>

template<class **T**>

struct **get_type**<int16_t, *T*> : public std::integral_constant<*type_t*, int16_type>

template<class **T**>

struct **get_type**<int32_t, *T*> : public std::integral_constant<*type_t*, int32_type>

template<class **T**>

struct **get_type**<int64_t, *T*> : public std::integral_constant<*type_t*, int64_type>

template<class **T**>

struct **get_type**<int8_t, *T*> : public std::integral_constant<*type_t*, int8_type>

template<class **T**>

struct **get_type**<migraphx::fp8::fp8e4m3fn, *T*> : public std::integral_constant<*type_t*, fp8e4m3fn_type>

template<class **T**>

struct **get_type**<migraphx::fp8::fp8e4m3fnuz, *T*> : public std::integral_constant<*type_t*, fp8e4m3fnuz_type>

template<class **T**>

struct **get_type**<migraphx::fp8::fp8e5m2, *T*> : public std::integral_constant<*type_t*, fp8e5m2_type>

template<class **T**>

struct **get_type**<migraphx::fp8::fp8e5m2fnuz, *T*> : public std::integral_constant<*type_t*, fp8e5m2fnuz_type>

template<class **T**>

struct **get_type**<uint16_t, *T*> : public std::integral_constant<*type_t*, uint16_type>

template<class **T**>

struct **get_type**<uint32_t, *T*> : public std::integral_constant<*type_t*, uint32_type>

template<class **T**>

struct **get_type**<uint64_t, *T*> : public std::integral_constant<*type_t*, uint64_type>

template<class **T**>

struct **get_type**<uint8_t, *T*> : public std::integral_constant<*type_t*, uint8_type>

5.4.2 literal

struct **literal** : public migraphx::internal::raw_data<*literal*>

Represents a raw literal.

This stores the literal has a raw buffer that is owned by this class

Public Functions

inline **literal**()

inline explicit **literal**(*shape*::type_t shape_type)

Empty literal with a specific shape type

template<class **U**, class **T** = deduce<*U*>, *shape*::type_t **ShapeType** = *shape*::get_type<*T*>{}>
inline **literal**(*U* x)

template<class **T**>
inline **literal**(const *shape* &s, const std::vector<*T*> &x)

template<class **T**>
inline **literal**(const *shape* &s, const std::initializer_list<*T*> &x)

template<class **Iterator**>
inline **literal**(const *shape* &s, *Iterator* start, *Iterator* end)

template<class **T**, long **PrivateRequires__LINE__** = __LINE__, typename std::enable_if<(*PrivateRequires__LINE__* == __LINE__ and (migraphx::and_<sizeof(*T*) == 1>{})), int>::type = 0>
inline **literal**(const *shape* &s, *T* *x)

inline bool **empty**() const

Whether data is available.

inline const char ***data**() const

Provides a raw pointer to the data.

inline const *shape* &**get_shape**() const

inline std::vector<*literal*> **get_sub_objects**() const

inline *argument* **get_argument**() const

Convert the data to an argument.

5.4.3 argument

struct **argument** : public migraphx::internal::raw_data<*argument*>

Arguments passed to instructions.

An `argument` can represent a raw buffer of data that either be referenced from another element or it can be owned by the argument.

Public Functions

argument() = default

explicit **argument**(const *shape* &s)

template<class **F**, long **PrivateRequires__LINE__** = __LINE__, typename std::enable_if<(*PrivateRequires__LINE__* == __LINE__ and (migraphx::and_<std::is_pointer<decltype(std::declval<*F*>()())>{}>{})), int>::type = 0>
inline **argument**(*shape* s, *F* d)

template<class **T**>
inline **argument**(*shape* s, *T* *d)

template<class **T**>
inline **argument**(*shape* s, const std::shared_ptr<*T*> &d)

argument(*shape* s, std::nullptr_t)

argument(const std::vector<*argument*> &args)

char ***data**() const

Provides a raw pointer to the data.

bool **empty**() const

Whether data is available.

const *shape* &**get_shape**() const

argument **reshape**(const *shape* &s) const

argument **copy**() const

argument **share**() const

Make copy of the argument that is always sharing the data.

std::vector<*argument*> **get_sub_objects**() const

argument **element**(std::size_t i) const

Return the ith element.

template<class **Iterator**>
inline void **fill**(*Iterator* start, *Iterator* end)

argument **convert**(*shape*::*type_t* t) const

5.4.4 raw_data

template<class **Derived**>

struct **raw_data** : public migraphx::internal::raw_data_base

Provides a base class for common operations with raw buffer.

For classes that handle a raw buffer of data, this will provide common operations such as equals, printing, and visitors. To use this class the derived class needs to provide a `data()` method to retrieve a raw pointer to the data, and `get_shape` method that provides the shape of the data.

Public Functions

template<class **Visitor**, class **Index** = std::size_t>

inline void **visit_at**(*Visitor* v, *Index* n = 0) const

Visits a single data element at a certain index.

Parameters

- **v** – A function which will be called with the type of data
- **n** – The index to read from

template<class **Visitor**, class **TupleVisitor**>
inline void **visit**(*Visitor* v, *TupleVisitor* tv) const

template<class **Visitor**>
inline void **visit**(*Visitor* v) const

Visits the data.

This will call the visitor function with a `tensor_view`<T> based on the shape of the data.

Parameters

v – A function to be called with `tensor_view`<T>

template<class **Visitor**, class **TupleVisitor**>
inline void **fallback_visit**(*Visitor* v, *TupleVisitor* tv) const

Visit the data using the normal visit function for computable types. For non-computable types, use a tensor_view<byte> with shape = {type = uint8_type, lens = {num bytes}};

template<class **Visitor**>
inline void **fallback_visit**(*Visitor* v) const

inline bool **single**() const

Returns true if the raw data is only one element.

template<class **T**, class **Index** = std::size_t>
inline *T* **at**(*Index* n = 0) const

Retrieves a single element of data.

Parameters

n – The index to retrieve the data from

Template Parameters

T – The type of data to be retrieved

Returns

The element as `T`

inline *auto_cast* **implicit**() const

Implicit conversion of raw data pointer.

template<class **T**>
inline tensor_view<*T*> **get**() const

Get a *tensor_view* to the data. For *get<byte>()* returns a 1D tensor_view<const byte*>.

template<class **T**>
inline *T* ***cast**() const

inline std::string **to_string**() const

template<class **T**>
inline std::vector<*T*> **to_vector**() const

Friends

template<class **Stream**>
inline friend *Stream* &**operator<<**(*Stream* &os, const *Derived* &d)

struct **auto_cast**

Public Types

template<class **T**>

using **is_data_ptr** = bool_c<(std::is_void<*T*>{} or std::is_same<char, std::remove_cv_t<*T*>>{} or std::is_same<unsigned char, std::remove_cv_t<*T*>>{})>

template<class **T**>

using **get_data_type** = std::conditional_t<*is_data_ptr*<*T*>{}, float, *T*>

Public Functions

template<class **T**>
inline **operator** *T*()

template<class **T**>
inline bool **matches**() const

template<class **T**>
inline **operator** *T****()

Public Members

const *Derived* ***self**

template<class **T**, class ...**Ts**>
auto `migraphx::internal::`**visit_all**(*T* &&x, *Ts*&&... xs)

Visits every object together.

This will visit every object, but assumes each object is the same type. This can reduce the deeply nested visit calls. Returns a function that takes the visitor callback. Calling syntax is `visit_all(xs...)([](auto... ys) {})` where `xs...` and `ys...` are the same number of parameters.

Parameters

- **x** – A raw data object
- **xs** – Many raw data objects.

Returns

A function to be called with the visitor

template<class **T**>
auto `migraphx::internal::`**visit_all**(const std::vector<*T*> &x)

Visits every object together.

This will visit every object, but assumes each object is the same type. This can reduce the deeply nested visit calls. Returns a function that takes the visitor callback.

Parameters

x – A vector of raw data objects. Types must all be the same.

Returns
A function to be called with the visitor

5.4.5 tensor_view

template<class **T**>

struct **tensor_view**

Public Types

using **value_type** = *T*

using **iterator** = basic_iota_iterator<tensor_view_iterator_read<*tensor_view*<*T*>>, std::size_t>

using **const_iterator** = basic_iota_iterator<tensor_view_iterator_read<const *tensor_view*<*T*>>, std::size_t>

Public Functions

inline **tensor_view**()

inline **tensor_view**(*shape* s, *T* *d)

inline const *shape* &**get_shape**() const

inline bool **empty**() const

inline std::size_t **size**() const

inline *T* ***data**()

inline const *T* ***data**() const

template<class ...**Ts**, long **PrivateRequires__LINE__** = __LINE__, typename std::enable_if<(*PrivateRequires__LINE__* == __LINE__ and (migraphx::and_<std::is_integral<*Ts*>{}...>{})), int>::type = 0>
inline const *T* &**operator()**(*Ts*... xs) const

template<class ...**Ts**, long **PrivateRequires__LINE__** = __LINE__, typename std::enable_if<(*PrivateRequires__LINE__* == __LINE__ and (migraphx::and_<std::is_integral<*Ts*>{}...>{})), int>::type = 0>
inline *T* &**operator()**(*Ts*... xs)

template<class **Iterator**, long **PrivateRequires__LINE__** = __LINE__, typename std::enable_if<(*PrivateRequires__LINE__* == __LINE__ and (migraphx::and_<not std::is_integral<*Iterator*>{}>{})), int>::type = 0>
inline const *T* &**operator()**(*Iterator* start, *Iterator* last) const

template<class **Iterator**, long **PrivateRequires__LINE__** = __LINE__, typename std::enable_if<(*PrivateRequires__LINE__* == __LINE__ and (migraphx::and_<not std::is_integral<*Iterator*>{}>{})), int>::type = 0>
inline *T* &**operator()**(*Iterator* start, *Iterator* last)

inline *T* &**operator[]**(std::size_t i) inline const *T* &**operator[]**(std::size_t i) const

template<class **Range**>
inline auto **operator[]**(const *Range* &r) -> decltype((*this)(*r*.begin(), *r*.end()))

template<class **Range**>
inline auto **operator[]**(const *Range* &r) const -> decltype((*this)(*r*.begin(), *r*.end()))

inline *T* &**front**()

inline const *T* &**front**() const

inline *T* &**back**()

inline const *T* &**back**() const

inline *iterator* **begin**()

template<class **Range**>
inline *iterator* **begin_at**(const *Range* &r)

inline *iterator* **end**()

inline *const_iterator* **begin**() const

template<class **Range**>
inline *const_iterator* **begin_at**(const *Range* &r) const

inline *const_iterator* **end**() const

template<class **U** = *T*>
inline std::vector<*U*> **to_vector**() const

template<class **Range**>
inline *tensor_view* **slice_at**(std::initializer_list<std::int64_t> axes, *Range* &&r)

Friends

inline friend std::ostream &**operator<<**(std::ostream &os, const *tensor_view*<*T*> &x)

5.5 Operators

5.5.1 operation

struct **operation**

The operation interface represents an action an instruction will perform. All operation classes must be CopyConstructible.

Public Functions

std::string **name**() const

A unique name identifying the operation.

void **finalize**(context &ctx)

An optional method that can be used to finalize the operator before running.

shape **compute_shape**(const std::vector<*shape*> &input) const

This is used to compute the resulting shape from an operation. If an operation cannot be run with input shapes, then it should throw an exception.

argument **compute**(context &ctx, const *shape* &output, const std::vector<*argument*> &input) const

This performs the operation's computation.

This method can be optional when the operation is only used as a placeholder to be lowered later on.

Parameters

- **ctx** – This is the context created by the `target` during compilation. Implementations can use the target's `context` class rather than the `context` interface class.
- **output** – Equivalent to running `compute_shape` with each `shape` of the `argument`. For a fixed shape, the returned argument will have the same shape as `output`. For a dynamic shape, the returned `argument` will be a fixed shape within the bounds set in the dynamic shape `output`.
- **input** – This is the `argument` result from the previous instruction's computation.

Returns

Return an `argument` of the result computation. The `shape` of `argument` should be the same the `output` shape.

std::vector<std::size_t> **output_alias**(const std::vector<*shape*> &input) const

An optional method to return which arguments the output will alias. If there is no aliased output then an empty vector can be returned.

Friends

friend std::ostream &**operator<<**(std::ostream &os, const *operation* &op)

An optional stream operator to print the operation. When this is not implemented, it will just print the operation's name.

bool `migraphx::internal::`**is_context_free**(const *operation* &x)

Returns true if operation does not require a context to run compute.

bool `migraphx::internal::`**has_finalize**(const *operation* &x)

Returns true if the operation has a finalize method.

5.5.2 operators

namespace **op**

Unpacks fastest dimension of tensor into fp8e4m3fn_type such that the output dimensions are [x_0, …, 2 * x_pack, …]

Enums

enum **padding_mode_t**

Values:

enumerator **default_**

enumerator **same_lower**

enumerator **same_upper**

enum class **pooling_mode**

Values:

enumerator **average**

enumerator **max**

enumerator **lpnorm**

enum class **rnn_direction**

Values:

enumerator **forward**

enumerator **reverse**

enumerator **bidirectional**

enum class **normalize_attribute**

`normalize_attribute` settings: Note that default options are not included as enums.

i. `use_input` (default) vs. `use_output`: Affects the rank of the attribute. `use_input -> lens.size()`, `use_output -> lens.size() + vec.size()`.
ii. use_rank (default) vs use_len: `use_rank` sets the max value/index of the attribute as the rank of lens. `use_lens` sets the max value/index as the corresponding value in lens at the axes index. Uses the dynamic_dimension.max value for dynamic shapes. Returns the original vector (no normalization) if any of dynamic_dimension[axes] are not fixed.
iii. `clip_min` vs. `not_clip_min` (default): Clip values less than the minimum to the minimum or not.
iv. `include_min` vs. `exclude_min` (default): Include or exclude the minimum value/index for range checking and clipping.
v. `clip_max` vs. `not_clip_max` (default): Clip values greater than the maximum or not.
vi. `include_max` vs. `exclude_max` (default): Include or exclude the maximum value/index for range checking and clipping.
vii. `normalize_padding`: To normalize the padding to `2*(pad ndim)` dimensions.

Values:

enumerator **use_output**

enumerator **use_len**

enumerator **clip_max**

enumerator **clip_min**

enumerator **include_max**

enumerator **include_min**

enumerator **normalize_padding**

Functions

std::ostream &**operator<<**(std::ostream &os, *pooling_mode* v)

std::ostream &**operator<<**(std::ostream &os, *rnn_direction* v)

struct **abs** : public migraphx::internal::op::unary<*abs*>

#include <migraphx/op/abs.hpp>

Public Functions

inline auto **apply**() const

struct **acos** : public migraphx::internal::op::unary<*acos*>

#include <migraphx/op/acos.hpp>

Public Functions

inline auto **apply**() const

struct **acosh** : public migraphx::internal::op::unary<*acosh*>

#include <migraphx/op/acosh.hpp>

Public Functions

inline auto **apply**() const

struct **add** : public migraphx::internal::op::binary<*add*>

#include <migraphx/op/add.hpp>

Public Functions

inline value **attributes**() const

inline std::string **point_function**() const

inline auto **apply**() const

struct **allocate**

#include <migraphx/op/allocate.hpp> Static allocate: No inputs: `allocate()` `this.s` attribute set to the static output shape of the buffer. `this.s` attribute can be set to a dynamic output shape; however this will allocate the maximum buffer size for that case

Dynamic allocate: One input: `allocate(output_dims)` `output_dims` are the output buffer dimensions and has a static shape. Either `this.s` or `this.buf_type` (but not both) must be set to calculate the dynamic output shape at compute time. If `this.buf_type` is set, the *compute_shape()* of allocate at compile time will have dynamic_dimensions from {0, max_int} with rank = output_dims.ndim(). If `this.s` is set then the *compute_shape()* will output `this.s`; `this.s` should be a dynamic shape.

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(const std::vector<*shape*> &inputs) const

inline *argument* **compute**(const *shape* &output_shape, const std::vector<*argument*> &args) const

Public Members

optional<*shape*> **s**

optional<*shape*::type_t> **buf_type**

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **argmax**

#include <migraphx/op/argmax.hpp>

Public Functions

inline value **attributes**() const

inline std::string **name**() const

inline *shape* **normalize_compute_shape**(std::vector<*shape*> inputs) const

template<class **T**>
inline int64_t **calc_argmax**(*T* &input, std::vector<std::size_t> &indices, size_t item_num) const

inline *argument* **compute**(const dyn_output &dyn_out, std::vector<*argument*> args) const

Public Members

int64_t **axis** = 0

bool **select_last_index** = false

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **argmin**

#include <migraphx/op/argmin.hpp>

Public Functions

inline value **attributes**() const

inline std::string **name**() const

inline *shape* **normalize_compute_shape**(std::vector<*shape*> inputs) const

template<class **T**>
inline int64_t **calc_argmin**(*T* &input, std::vector<std::size_t> &indices, size_t item_num) const

inline *argument* **compute**(const *shape* &output_shape, std::vector<*argument*> args) const

Public Members

int64_t **axis** = 0

bool **select_last_index** = false

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **as_shape**

#include <migraphx/op/as_shape.hpp>

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(const std::vector<*shape*> &inputs) const

inline *argument* **compute**(*shape* output_shape, std::vector<*argument*> args) const

inline std::vector<std::size_t> **output_alias**(const std::vector<*shape*>&) const

Public Members

shape **s**

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **asin** : public migraphx::internal::op::unary<*asin*>

#include <migraphx/op/asin.hpp>

Public Functions

inline auto **apply**() const

struct **asinh** : public migraphx::internal::op::unary<*asinh*>

#include <migraphx/op/asinh.hpp>

Public Functions

inline auto **apply**() const

struct **atan** : public migraphx::internal::op::unary<*atan*>

#include <migraphx/op/atan.hpp>

Public Functions

inline auto **apply**() const

struct **atanh** : public migraphx::internal::op::unary<*atanh*>

#include <migraphx/op/atanh.hpp>

Public Functions

inline auto **apply**() const

template<class **Derived**>

struct **binary** : public migraphx::internal::op::op_name<*Derived*>

#include <migraphx/op/binary.hpp>

Public Functions

inline std::string **point_function**() const

inline std::string **point_op**() const

inline value **base_attributes**() const

inline value **attributes**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const dyn_output &dyn_out, std::vector<*argument*> args) const

struct **bit_cast** : public migraphx::internal::op::unary<*bit_cast*>

#include <migraphx/op/bit_cast.hpp> Obtain a value of type `target_type` by reinterpreting the object represnetaion of the input. Originally used for casting from fp8e4m3fn to fp8e4m3fnuz.

Public Functions

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

inline std::string **point_op**() const

inline *argument* **compute**(const dyn_output &dyn_out, std::vector<*argument*> args) const

Public Members

shape::type_t **target_type**

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **bitwise_and** : public migraphx::internal::op::binary<*bitwise_and*>

#include <migraphx/op/bitwise_and.hpp>

Public Functions

inline value **attributes**() const

inline std::string **point_function**() const

inline *shape* **compute_shape**(const std::vector<*shape*> &inputs) const

inline auto **apply**() const

struct **broadcast**

#include <migraphx/op/broadcast.hpp> 1 input version: Broadcasts a tensor from the original shape to the broadcast_lens by setting the stride of broadcasted dimensions to zero. `axis` attribute for a 1D input shape is the output dimension that stays the same. ex: broadcasting shape [1024] -> [4, 1024, 3] has axis = 1.

For higher rank input shapes, axis is an offset parameter for the broadcasting. Such that this operator would work in the opposite direction of NumPy broadcasting (left-most to rightwards element-wise comparison) ex: broadcasting shape [2, 2] -> [2, 2, 3] with axis = 0

2 input version: Broadcast the first input 1D shape into the second input shape based on the axis parameter. Handles broadcasting a 1D static shape into a higher rank dynamic shape. broadcast_lens is not used

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const dyn_output &dyn_out, std::vector<*argument*> args) const

inline std::vector<std::size_t> **output_alias**(const std::vector<*shape*>&) const

inline value **attributes**() const

Public Members

uint64_t **axis** = 0

std::vector<std::size_t> **broadcast_lens** = {}

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **broadcast_for_dot**

#include <migraphx/op/broadcast_for_dot.hpp> Broadcast dimensions between two tensors for the `dot` operator. Essentially broadcasts between two shapes for dimensions other than the last two. This operator is only needed if one of the shapes are dynamic. Example: a = shape[{1, 4}, 3, 248, 248] b = shape[248, 365] broadcast_for_dot(a, b) => shape[{1, 4}, 3, 248, 248] (no change) broadcast_for_dot(b, a) => shape[{1, 4}, 3, 248, 365]

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const dyn_output &dyn_out, std::vector<*argument*> args) const

inline std::vector<std::size_t> **output_alias**(const std::vector<*shape*>&) const

inline value **attributes**() const

struct **broadcast_with_dims**

#include <migraphx/op/broadcast_with_dims.hpp> Broadcast the input tensor to the shape defined by the values of the second input. Used as `broadcast_with_dims(input_tensor, dims)`, where dims is a vector of integer dimensions. `input_tensor` must be broadcastable with `dims`, otherwise this operator with throw at compute. This operator can be replaced with `multibroadcast(input_tensor)` if the `dims` vector is constant.

Example: input_tensor shape: lens = {2, 3}, strides = {3, 1} dims = [4, 1, 3] output shape: lens = {4, 2, 3}, strides = {0, 3, 1}

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(const std::vector<*shape*> &inputs) const

inline *argument* **compute**(const *shape* &output_shape, const std::vector<*argument*> &args) const

inline value **attributes**() const

struct **cache_parameters**

#include <migraphx/op/concat_past_present.hpp>

Public Members

std::size_t **batch_size** = 0

std::size_t **sequence_length** = 0

std::size_t **head_size** = 0

std::size_t **seqlen_present_kv_cache** = 0

struct **capture**

#include <migraphx/op/capture.hpp>

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(context&, const *shape*&, const std::vector<*argument*> &args) const

inline std::vector<std::size_t> **output_alias**(const std::vector<*shape*>&) const

Public Members

std::size_t **ins_index**

std::function<void(std::size_t ins_index, std::vector<*argument*>)> **f** = {}

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **ceil** : public migraphx::internal::op::unary<*ceil*>

#include <migraphx/op/ceil.hpp>

Public Functions

inline auto **apply**() const

struct **clip**

#include <migraphx/op/clip.hpp>

Public Functions

inline std::string **name**() const

inline value **attributes**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const dyn_output &dyn_out, std::vector<*argument*> args) const

struct **concat**

#include <migraphx/op/concat.hpp>

Public Functions

inline value **attributes**() const

inline std::string **name**() const

inline std::vector<std::size_t> **compute_offsets**(const *shape* &output_shape, const std::vector<*argument*> &args) const

inline *shape* **normalize_compute_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const dyn_output &dyn_out, std::vector<*argument*> args) const

Public Members

int64_t **axis** = 0

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **concat_past_present**

#include <migraphx/op/concat_past_present.hpp>

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

template<class **T**>
inline void **copy_data**(*T* destination, const *T* source, std::size_t n) const

template<typename **T**>
inline *T* **concat_state_chunk**(const *T* chunk, const *T* present, std::size_t present_buff_chunk_length, std::size_t past_chunk_length, std::size_t new_chunk_length, std::ptrdiff_t i) const

template<class **T**, class **U**>
inline void **update_cache**(*T* past_key, const *U* seqlens_k, const *T* present_key, *cache_parameters* params) const

inline *argument* **compute**(const *shape*&, std::vector<*argument*> args) const

Public Members

std::size_t **kv_num_heads** = 0

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **contiguous**

#include <migraphx/op/contiguous.hpp> The contiguous operator takes a non-standard input tensor and returns the same tensor but in standard form. For example, if input tensor A which has lens = (4,5) is first transposed, i.e. lens = (5,4), this tensor's data layout remained the same during the transpose operation; only it's shape lengths and strides were changed. This leaves the tensor in a non-standard form. The contiguous operator copies the underlying data such that resulting tensor is returned to a standard form.

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const dyn_output &dyn_out, std::vector<*argument*> args) const

inline auto **apply**() const

struct **convert** : public migraphx::internal::op::unary<*convert*>

#include <migraphx/op/convert.hpp>

Public Functions

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

inline std::string **point_op**() const

inline auto **apply**() const

inline **convert**(*shape*::type_t t)

inline **convert**()

Public Members

shape::type_t **target_type** = *shape*::half_type

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **convolution**

#include <migraphx/op/convolution.hpp> Convolution operator. Does not support optimal dimensions for spatial dimensions. Returns empty optimals.

Public Functions

inline std::string **name**() const

inline void **check_attribute_size**() const

inline value **attributes**() const

inline *shape* **normalize_compute_shape**(std::vector<*shape*> inputs) const

inline std::vector<std::size_t> **calc_conv_lens**(std::vector<std::size_t> x_lens, std::vector<std::size_t> w_lens) const

inline *shape* **dynamic_compute_shape**(*shape* x_shape, *shape* w_shape) const

inline *shape* **static_compute_shape**(*shape* x_shape, *shape* w_shape) const

inline size_t **kdims**() const

inline *argument* **compute**(*shape* output_shape, std::vector<*argument*> args) const

Public Members

std::vector<std::size_t> **padding** = {0, 0}

std::vector<std::size_t> **stride** = {1, 1}

std::vector<std::size_t> **dilation** = {1, 1}

int **group** = 1

padding_mode_t **padding_mode** = default_

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **convolution_backwards**

#include <migraphx/op/convolution_backwards.hpp>

Public Functions

inline std::string **name**() const

inline void **check_attribute_size**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

inline std::vector<std::size_t> **calc_spatial_lens**(std::vector<std::size_t> x_lens, std::vector<std::size_t> w_lens) const

inline *shape* **dynamic_compute_shape**(*shape* x_shape, *shape* w_shape) const inline *shape* **static_compute_shape**(*shape* x_shape, *shape* w_shape) const

inline *argument* **compute**(const dyn_output &dyn_out, std::vector<*argument*> args) const

inline size_t **kdims**() const

Public Members

std::vector<std::size_t> **padding** = {0, 0}

std::vector<std::size_t> **stride** = {1, 1}

std::vector<std::size_t> **dilation** = {1, 1}

padding_mode_t **padding_mode** = default_

int **group** = 1

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **cos** : public migraphx::internal::op::unary<*cos*>

#include <migraphx/op/cos.hpp>

Public Functions

inline auto **apply**() const

struct **cosh** : public migraphx::internal::op::unary<*cosh*>

#include <migraphx/op/cosh.hpp>

Public Functions

inline auto **apply**() const

struct **dequantizelinear**

#include <migraphx/op/dequantizelinear.hpp>

Public Functions

inline value **attributes**() const

inline std::string **name**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const *shape* &output_shape, std::vector<*argument*> args) const struct **dimensions_of**

#include <migraphx/op/dimensions_of.hpp> Returns the dimensions of the input argument from starting axis to ending axis. Atleast end must be set to use this operator (set end to ndim for default ONNX behavior of Shape operator) This should only be used for dynamic shapes as this can be simplified to a literal for static shapes.

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(const std::vector<*shape*> &inputs) const

inline *argument* **compute**(const *shape* &output_shape, std::vector<*argument*> args) const

Public Members

std::size_t **start** = 0

std::size_t **end** = 0

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **div** : public migraphx::internal::op::binary<*div*>

#include <migraphx/op/div.hpp>

Public Functions

inline std::string **point_function**() const

inline auto **apply**() const

struct **dot**

#include <migraphx/op/dot.hpp> Matrix multiplication of two tensors.

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const dyn_output &dyn_out, std::vector<*argument*> args) const

struct **elu** : public migraphx::internal::op::unary<*elu*>

#include <migraphx/op/elu.hpp>

Public Functions

inline std::string **point_op**() const

inline auto **apply**() const

Public Members

float **alpha** = 1

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **equal** : public migraphx::internal::op::binary<*equal*>

#include <migraphx/op/equal.hpp>

Public Functions

inline value **attributes**() const

inline std::string **point_function**() const

inline auto **apply**() const

struct **erf** : public migraphx::internal::op::unary<*erf*>

#include <migraphx/op/erf.hpp>

Public Functions

inline auto **apply**() const

struct **exp** : public migraphx::internal::op::unary<*exp*>

#include <migraphx/op/exp.hpp>

Public Functions

inline auto **apply**() const

struct **fill**

#include <migraphx/op/fill.hpp> fill(default_value, output_buffer) Fill an output buffer with the given default_value. Note that if the default_value is a literal and the output_buffer has a static shape this operator can be replaced with a literal.

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const dyn_output &dyn_out, std::vector<*argument*> args) const inline std::vector<std::size_t> **output_alias**(const std::vector<*shape*>&) const

struct **fixed_pad**

#include <migraphx/op/fixed_pad.hpp> Pads an input with dynamic shape to its maximum dimensions. No-op for a static shape input. The main use for this op versus the standard pad op is that it can accept a dynamic input shape and convert it to a padded static shape.

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const *shape* &output_shape, std::vector<*argument*> args) const

struct **flatten**

#include <migraphx/op/flatten.hpp>

Public Functions

inline value **attributes**() const

inline std::string **name**() const

inline *shape* **normalize_compute_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const dyn_output &dyn_out, std::vector<*argument*> args) const

Public Members

int64_t **axis** = 1

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **floor** : public migraphx::internal::op::unary<*floor*>

#include <migraphx/op/floor.hpp>

Public Functions

inline auto **apply**() const

struct **fmod** : public migraphx::internal::op::binary<*fmod*>

#include <migraphx/op/fmod.hpp>

Public Functions

inline std::string **name**() const

inline value **attributes**() const

inline auto **apply**() const

struct **gather**

#include <migraphx/op/gather.hpp>

Public Functions

inline value **attributes**() const

inline std::string **name**() const

inline *shape* **normalize_compute_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const dyn_output &dyn_out, std::vector<*argument*> args) const

Public Members

int64_t **axis** = 0

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **gathernd**

#include <migraphx/op/gathernd.hpp>

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const dyn_output &dyn_out, std::vector<*argument*> args) const

Public Members

int **batch_dims** = 0

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **get_tuple_elem**

#include <migraphx/op/get_tuple_elem.hpp>

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const *shape*&, std::vector<*argument*> args) const

inline std::vector<std::size_t> **output_alias**(const std::vector<*shape*>&) const

Public Members

std::size_t **index** = 0

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **gqa_rotary_embedding**

#include <migraphx/op/gqa_rotary_embedding.hpp>

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

template<class **T**>
inline void **run_rotary_embedding**(*T* input, *T* cos_cache, *T* sin_cache, *T* output, const size_t *pos_ids, *rotary_parameters* params) const

template<class **T**>
inline void **pack_v_into_rotary_qkv**(*rotary_parameters* params, const *T* input, *T* output) const

inline *argument* **compute**(const *shape* &output_shape, std::vector<*argument*> args) const

Public Members

size_t **num_heads** = 1

size_t **kv_num_heads** = 1

bool **interleaved** = false

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **rotary_parameters**

#include <migraphx/op/gqa_rotary_embedding.hpp>

Public Members

size_t **batch_size** = 0

size_t **sequence_length** = 0

size_t **head_size** = 0

size_t **num_heads** = 0

size_t **rotary_embedding_dim** = 0

size_t **max_sequence_length** = 0

size_t **head_stride** = 0

size_t **seq_stride** = 0

size_t **batch_stride** = 0

bool **position_ids_use_batch** = false

struct **greater** : public migraphx::internal::op::binary<*greater*>

#include <migraphx/op/greater.hpp>

Public Functions

inline std::string **point_function**() const

inline auto **apply**() const

struct **group**

#include <migraphx/op/group.hpp>

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(const std::vector<*shape*> &inputs, const std::vector<module_ref> &mods) const

Public Members

std::string **tag** = ""

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **gru**

#include <migraphx/op/gru.hpp>

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

Public Members

std::size_t **hidden_size** = 1

std::vector<operation> **actv_funcs** = {*sigmoid*{}, *tanh*{}}

rnn_direction **direction** = *rnn_direction*::*forward*

float **clip** = 0.0f

int **linear_before_reset** = 0

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **highest**

#include <migraphx/op/reduce_op.hpp>

Public Functions

template<class **T**>
inline **operator** *T*() const

struct **identity**

#include <migraphx/op/identity.hpp>

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(*shape*, std::vector<*argument*> args) const

inline value **attributes**() const

inline std::vector<std::size_t> **output_alias**(const std::vector<*shape*>&) const

struct **if_op**

#include <migraphx/op/if_op.hpp>

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(const std::vector<*shape*> &inputs, std::vector<module_ref> mods) const

inline *argument* **compute**(const *shape*&, const std::vector<*argument*> &args, const std::vector<module_ref> &mods, const std::function<std::vector<*argument*>(module_ref&, const std::unordered_map<std::string, *argument*>&)> &run) const

struct **im2col**

#include <migraphx/op/im2col.hpp>

Public Functions

inline std::string **name**() const

inline value **attributes**() const

inline *shape* **normalize_compute_shape**(std::vector<*shape*> inputs) const

Public Members

std::vector<std::size_t> **padding** = {0, 0}

std::vector<std::size_t> **stride** = {1, 1}

std::vector<std::size_t> **dilation** = {1, 1}

padding_mode_t **padding_mode** = default_

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **isinf** : public migraphx::internal::op::unary<*isinf*>

#include <migraphx/op/isinf.hpp>

Public Functions

inline auto **apply**() const

inline std::string **name**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

struct **isnan** : public migraphx::internal::op::unary<*isnan*>

#include <migraphx/op/isnan.hpp>

Public Functions

inline auto **apply**() const

inline std::string **name**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

struct **layout** : public migraphx::internal::op::unary<*layout*>

#include <migraphx/op/layout.hpp> Rearrange the memory layout of the input instruction based on the permutation attribute. This operator changes the order of elements in memory, *not* the order in the tensor. Therefore, regardless of how the memory layout is changed, the order of elements returned by a *tensor_view* will be unchanged. `permutation`: List with how to rearrange the data buffer of the input instruction. This permutation is the transpose from the order in the tensor to the order in memory.

Public Functions

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

inline auto **apply**() const

Public Members

std::vector<int64_t> **permutation**

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **leaky_relu** : public migraphx::internal::op::unary<*leaky_relu*>

#include <migraphx/op/leaky_relu.hpp>

Public Functions

inline std::string **point_op**() const

inline std::string **name**() const

inline auto **apply**() const

Public Members

float **alpha** = 0.01

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **less** : public migraphx::internal::op::binary<*less*>

#include <migraphx/op/less.hpp>

Public Functions

inline std::string **point_function**() const

inline auto **apply**() const

struct **load**

#include <migraphx/op/load.hpp>

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(const std::vector<*shape*> &inputs) const

inline *argument* **compute**(const *shape*&, const std::vector<*argument*> &args) const

inline lifetime **get_lifetime**() const

inline std::vector<std::size_t> **output_alias**(const std::vector<*shape*>&) const

Public Members

shape **s**

std::size_t **offset** = 0

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

Friends

inline friend std::ostream &**operator<<**(std::ostream &os, const *load* &op)

struct **log** : public migraphx::internal::op::unary<*log*>

#include <migraphx/op/log.hpp>

Public Functions

inline auto **apply**() const

struct **log2** : public migraphx::internal::op::unary<*log2*>

#include <migraphx/op/log2.hpp>

Public Functions

inline auto **apply**() const

struct **logical_and** : public migraphx::internal::op::binary<*logical_and*>

#include <migraphx/op/logical_and.hpp>

Public Functions

inline std::string **point_function**() const

inline auto **apply**() const

struct **logical_or** : public migraphx::internal::op::binary<*logical_or*>

#include <migraphx/op/logical_or.hpp>

Public Functions

inline std::string **point_function**() const

inline auto **apply**() const

struct **logical_xor** : public migraphx::internal::op::binary<*logical_xor*>

#include <migraphx/op/logical_xor.hpp>

Public Functions

inline std::string **point_function**() const

inline auto **apply**() const

struct **logsoftmax**

#include <migraphx/op/logsoftmax.hpp>

Public Functions

inline value **attributes**() const

inline std::string **name**() const

inline *shape* **normalize_compute_shape**(std::vector<*shape*> inputs) const

inline auto **output**() const

Public Members

int64_t **axis** = 1

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **loop**

#include <migraphx/op/loop.hpp>

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(const std::vector<*shape*> &inputs, std::vector<module_ref> mods) const

inline *argument* **compute**(context &ctx, const *shape* &out_shape, const std::vector<*argument*> &args, const std::vector<module_ref> &mods, const std::function<std::vector<*argument*>(module_ref&, const std::unordered_map<std::string, *argument*>&)> &run) const

Public Members

int64_t **max_iterations** = 10

std::vector<int64_t> **scan_output_directions** = { }

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **ref_loop**

#include <migraphx/op/loop.hpp>

Public Functions

template<class **T**>
inline void **copy**(context&, const *argument* &src, *T* &dst) const

template<class **T**>
inline void **copy**(context&, *T* src, const *argument* &dst) const

inline void **append**(const std::vector<*argument*> &iter_state, const std::vector<*argument*> &concatenated_outputs, const std::vector<int64_t> &scan_output_dirs, int64_t curr_iter, int64_t num_iters) const

inline void **set_zero**(context&, const std::vector<*argument*> &concatenated_outputs, int iter) const

inline std::unordered_map<std::string, int> **get_output_params**(const module&) const

Public Members

int64_t **max_iterations** = 0

struct **lowest**

#include <migraphx/op/reduce_op.hpp>

Public Functions

template<class **T**>
inline **operator** *T*() const

struct **lrn**

#include <migraphx/op/lrn.hpp>

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

Public Members

float **alpha** = 0.0001

float **beta** = 0.75

float **bias** = 1.0

int **size** = 1

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **lstm**

#include <migraphx/op/lstm.hpp>

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

Public Members

std::size_t **hidden_size** = 1

std::vector<operation> **actv_funcs** = {*sigmoid*{}, *tanh*{}, *tanh*{}}

rnn_direction **direction** = *rnn_direction*::*forward*

float **clip** = 0.0f

int **input_forget** = 0

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **max** : public migraphx::internal::op::binary<*max*>

#include <migraphx/op/max.hpp>

Public Functions

inline value **attributes**() const

inline auto **apply**() const

struct **min** : public migraphx::internal::op::binary<*min*>

#include <migraphx/op/min.hpp>

Public Functions

inline value **attributes**() const

inline auto **apply**() const

struct **mod** : public migraphx::internal::op::binary<*mod*>

#include <migraphx/op/mod.hpp>

Public Functions

inline std::string **name**() const

inline value **attributes**() const

inline auto **apply**() const

struct **mul** : public migraphx::internal::op::binary<*mul*>

#include <migraphx/op/mul.hpp>

Public Functions

inline value **attributes**() const

inline std::string **point_function**() const

inline auto **apply**() const

struct **multibroadcast**

#include <migraphx/op/multibroadcast.hpp> Broadcast multiple dimensions between two tensors. Two versions of this operator: 1 input and 2+ inputs. One input version uses output_lens attribute and broadcasts to it. 2+ inputs version broadcasts first input to the common shape at evaluation time.

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const dyn_output &dyn_out, std::vector<*argument*> args) const

inline std::vector<std::size_t> **output_alias**(const std::vector<*shape*>&) const

Public Members

std::vector<std::size_t> **output_lens** = {}

std::vector<*shape*::dynamic_dimension> **output_dyn_dims** = {}

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **multinomial**

#include <migraphx/op/multinomial.hpp>

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const dyn_output &dyn_out, std::vector<*argument*> args) const

Public Members

shape::type_t **dtype** = *shape*::type_t::int32_type

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **nearbyint** : public migraphx::internal::op::unary<*nearbyint*>

#include <migraphx/op/nearbyint.hpp>

Public Functions

inline auto **apply**() const

struct **neg** : public migraphx::internal::op::unary<*neg*>

#include <migraphx/op/neg.hpp>

Public Functions

inline std::string **point_function**() const

inline auto **apply**() const

struct **nonmaxsuppression**

#include <migraphx/op/nonmaxsuppression.hpp>

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

template<class **T**>
inline *box* **batch_box**(*T* boxes, std::size_t box_idx) const

inline bool **suppress_by_iou**(*box* b1, *box* b2, double iou_threshold) const

template<class **T**>
inline std::vector<std::pair<double, int64_t>> **filter_boxes_by_score**(*T* scores_start, std::size_t num_boxes, double score_threshold) const

template<class **Output**, class **Boxes**, class **Scores**>
inline std::size_t **compute_nms**(*Output* output, const *Boxes* &boxes, const *Scores* &scores, std::size_t max_output_boxes_per_class, double iou_threshold, double score_threshold) const

inline *argument* **compute**(const *shape* &output_shape, std::vector<*argument*> args) const

Public Members

bool **center_point_box** = false

bool **use_dyn_output** = false

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **box**

#include <migraphx/op/nonmaxsuppression.hpp>

Public Functions

inline void **sort**()

inline std::array<double, 2> &**operator[]**(std::size_t i)

inline double **area**() const

Public Members

std::array<double, 2> **x**

std::array<double, 2> **y**

struct **nonzero**

#include <migraphx/op/nonzero.hpp>

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const *shape* &output_shape, std::vector<*argument*> args) const

struct **one**

#include <migraphx/op/reduce_op.hpp>

Public Functions

template<class **T**>
inline **operator** *T*() const

struct **onehot**

#include <migraphx/op/onehot.hpp> Produces a one-hot tensor. Called with `axis` attribute that defaults to the last output axis Constant depth: `onehot(indices, values), depth attribute must be set; Variable depth:`onehot(indices, depth, values)`; indiciesas a N rank tensor of indices where value ison_value` depth`scalar with the number of classes for the one-hot dimension` values[off_value, on_value] `axiswhich axis to add the one-hot dimension to For axis = 0 and rank(indices) = 2:  output is A[indicies[j, k], j, k] = on_value; A[i, j, k] = off_value otherwise Can be simplified to other operators when`indices`has a static shape and` depth` is constant at compile-time.

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(const std::vector<*shape*> &inputs) const

inline *argument* **compute**(const *shape*&, std::vector<*argument*> args) const

Public Members

int64_t **axis** = -1

std::optional<int64_t> **depth**

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

template<class **Derived**>

struct **op_name**

#include <migraphx/op/name.hpp> Create name from class.

Subclassed by *migraphx::internal::op::binary< add >*, *migraphx::internal::op::binary< bitwise_and >*, *migraphx::internal::op::binary< div >*, *migraphx::internal::op::binary< equal >*, *migraphx::internal::op::binary< fmod >*, *migraphx::internal::op::binary< greater >*, *migraphx::internal::op::binary< less >*, *migraphx::internal::op::binary< logical_and >*, *migraphx::internal::op::binary< logical_or >*, *migraphx::internal::op::binary< logical_xor >*, *migraphx::internal::op::binary< max >*, *migraphx::internal::op::binary< min >*, *migraphx::internal::op::binary< mod >*, *migraphx::internal::op::binary< mul >*, *migraphx::internal::op::binary< pow >*, *migraphx::internal::op::binary< prelu >*, *migraphx::internal::op::binary< sqdiff >*, *migraphx::internal::op::binary< sub >*, *migraphx::internal::op::prefix_scan_op< prefix_scan_sum >*, *migraphx::internal::op::reduce_op< reduce_all >*, *migraphx::internal::op::reduce_op< reduce_any >*, *migraphx::internal::op::reduce_op< reduce_max >*, *migraphx::internal::op::reduce_op< reduce_mean >*, *migraphx::internal::op::reduce_op< reduce_min >*, *migraphx::internal::op::reduce_op< reduce_prod >*, *migraphx::internal::op::reduce_op< reduce_sum >*, *migraphx::internal::op::scatter_op< scatter_add >*, *migraphx::internal::op::scatter_op< scatter_max >*, *migraphx::internal::op::scatter_op< scatter_min >*, *migraphx::internal::op::scatter_op< scatter_mul >*, *migraphx::internal::op::scatter_op< scatter_none >*, *migraphx::internal::op::scatternd_op< scatternd_add >*, *migraphx::internal::op::scatternd_op< scatternd_max >*, *migraphx::internal::op::scatternd_op< scatternd_min >*, *migraphx::internal::op::scatternd_op< scatternd_mul >*, *migraphx::internal::op::scatternd_op< scatternd_none >*, *migraphx::internal::op::unary< abs >*, *migraphx::internal::op::unary< acos >*, *migraphx::internal::op::unary< acosh >*, *migraphx::internal::op::unary< asin >*, *migraphx::internal::op::unary< asinh >*, *migraphx::internal::op::unary< atan >*, *migraphx::internal::op::unary< atanh >*, *migraphx::internal::op::unary< bit_cast >*, *migraphx::internal::op::unary< ceil >*, *migraphx::internal::op::unary< convert >*, *migraphx::internal::op::unary< cos >*, *migraphx::internal::op::unary< cosh >*, *migraphx::internal::op::unary< elu >*, *migraphx::internal::op::unary< erf >*, *migraphx::internal::op::unary< exp >*, *migraphx::internal::op::unary< floor >*, *migraphx::internal::op::unary< isinf >*, *migraphx::internal::op::unary< isnan >*, *migraphx::internal::op::unary< layout >*, *migraphx::internal::op::unary< leaky_relu >*, *migraphx::internal::op::unary< log >*, *migraphx::internal::op::unary< log2 >*, *migraphx::internal::op::unary< nearbyint >*, *migraphx::internal::op::unary< neg >*, *migraphx::internal::op::unary< recip >*, *migraphx::internal::op::unary< relu >*, *migraphx::internal::op::unary< rsqrt >*, *migraphx::internal::op::unary< sigmoid >*, *migraphx::internal::op::unary< sign >*, *migraphx::internal::op::unary< sin >*, *migraphx::internal::op::unary< sinh >*, *migraphx::internal::op::unary< sqrt >*, *migraphx::internal::op::unary< tan >*, *migraphx::internal::op::unary< tanh >*, *migraphx::internal::op::unary< unary_not >*, *migraphx::internal::op::binary< Derived >*, *migraphx::internal::op::prefix_scan_op< Derived >*, *migraphx::internal::op::reduce_op< Derived >*, *migraphx::internal::op::scatter_op< Derived >*, *migraphx::internal::op::scatternd_op< Derived >*, *migraphx::internal::op::unary< Derived >*

Public Functions

inline std::string **name**() const

struct **outline**

#include <migraphx/op/outline.hpp>

Public Functions

inline *shape* **compute_shape**(const std::vector<*shape*> &inputs) const

inline *argument* **compute**(const *shape*&, const std::vector<*argument*>&) const

Public Members

shape **s**

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **pack_fp4**

#include <migraphx/op/pack_fp4.hpp> Packs fastest dimension of tensor into fp4x2_type such that the output dimensions are [x_0, …, x_pack/2, …]

Public Functions

inline std::string **name**() const

inline migraphx::*shape* **normalize_compute_shape**(std::vector<migraphx::*shape*> inputs) const

inline *argument* **compute**(const *shape* &output_shape, const std::vector<*argument*> &args) const

struct **pack_int4**

#include <migraphx/op/pack_int4.hpp>

Public Functions

inline std::string **name**() const

inline value **attributes**() const

inline migraphx::*shape* **normalize_compute_shape**(std::vector<migraphx::*shape*> inputs) const

inline *argument* **compute**(const *shape* &output_shape, std::vector<*argument*> args) const

Public Members

int64_t **axis** = -1

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **pad**

#include <migraphx/op/pad.hpp>

Public Types

enum **pad_op_mode_t**

Values:

enumerator **constant_pad**

enumerator **reflect_pad**

enumerator **edge_pad**

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

inline std::size_t **pad_ndims**() const

inline bool **symmetric**() const

Public Members

std::vector<int64_t> **pads**

float **value** = 0.0f

pad_op_mode_t **mode** = constant_pad

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **pointwise**

#include <migraphx/op/pointwise.hpp>

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(const std::vector<*shape*> &inputs, std::vector<module_ref> mods) const

inline *argument* **compute**(const *shape* &output_shape, const std::vector<*argument*> &args, const std::vector<module_ref> &mods, const std::function<std::vector<*argument*>(module_ref&, const std::unordered_map<std::string, *argument*>&)> &run) const

inline value **attributes**() const

Public Static Functions

static inline std::vector<*shape*> **remove_broadcasts**(const std::vector<*shape*> &inputs)

struct **pooling**

#include <migraphx/op/pooling.hpp>

Public Functions

inline std::string **name**() const

inline void **check_attribute_size**() const

inline size_t **kdims**() const

inline value **attributes**() const

inline std::size_t **dilate_dim**(std::size_t dim, std::size_t dilation) const

inline std::vector<std::size_t> **calc_spatial_dim_out**(const std::vector<std::size_t> &input_lens, std::size_t kdims) const

inline *shape* **normalize_compute_shape**(std::vector<*shape*> inputs) const

template<class **Type**, class **Out**, class **In**, class **Op**>
inline void **calc_pooling**(const *shape* &output_shape, *Out* &output, const *In* &input, const std::vector<std::size_t> &kernel_dims, const std::vector<std::size_t> &padding_vals, *Op* op) const

inline *argument* **compute**(const dyn_output &dyn_out, std::vector<*argument*> args) const

Public Members

pooling_mode **mode** = {*pooling_mode*::*average*}

std::vector<std::size_t> **padding** = {0, 0}

std::vector<std::size_t> **stride** = {1, 1}

std::vector<std::size_t> **lengths** = {1, 1} std::vector<std::size_t> **dilations** = {1, 1}

bool **ceil_mode** = false

int **lp_order** = 2

padding_mode_t **padding_mode** = *padding_mode_t*::*default_*

bool **dyn_global** = false

bool **count_include_pad** = false

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **avg_pool**

#include <migraphx/op/pooling.hpp>

Public Functions

template<class **T**>
inline double **init**() const

inline double **operator()**(double x, double y) const

inline double **final**(double x, std::size_t y) const

struct **lpnorm_pool**

#include <migraphx/op/pooling.hpp>

Public Functions

lpnorm_pool() = delete

inline explicit **lpnorm_pool**(int x)

template<class **T**>
inline double **init**() const

inline double **operator()**(double x, double y) const

inline double **final**(double x, std::size_t) const

Public Members

int **p** = 0

struct **max_pool**

#include <migraphx/op/pooling.hpp>

Public Functions

template<class **T**>
inline *T* **init**() const

inline double **operator()**(double x, double y) const

inline double **final**(double x, std::size_t) const

struct **pow** : public migraphx::internal::op::binary<*pow*>

#include <migraphx/op/pow.hpp>

Public Functions

inline auto **apply**() const

template<class **Derived**>

struct **prefix_scan_op** : public migraphx::internal::op::op_name<*Derived*>

#include <migraphx/op/prefix_scan_op.hpp> Parent struct for prefix scan operations. A prefix scan is equivalent to the C++ std::exclusive_scan or std::inclusive_scan. Given a list of numbers, a prefix scan sum op returns an equal size list of running totals of the values. Other operations besides addition can be supported by their own child ops.

Public Functions

inline value **attributes**() const

inline *shape* **normalize_compute_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const dyn_output &dyn_out, std::vector<*argument*> args) const

inline auto **init**() const

inline **prefix_scan_op**()

inline **prefix_scan_op**(int64_t ax)

inline **prefix_scan_op**(int64_t ax, bool excl)

inline **prefix_scan_op**(int64_t ax, bool excl, bool rev)

Public Members

int64_t **axis**

bool **exclusive** = false

bool **reverse** = false

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **prefix_scan_sum** : public migraphx::internal::op::prefix_scan_op<*prefix_scan_sum*>

#include <migraphx/op/prefix_scan_sum.hpp>

Public Functions

inline **prefix_scan_sum**()

inline **prefix_scan_sum**(int64_t ax)

inline **prefix_scan_sum**(int64_t ax, bool excl)

inline **prefix_scan_sum**(int64_t ax, bool excl, bool rev)

inline auto **op**() const

struct **prelu** : public migraphx::internal::op::binary<*prelu*>

#include <migraphx/op/prelu.hpp>

Public Functions

inline std::string **point_op**() const

inline auto **apply**() const

struct **quant_convolution**

#include <migraphx/op/quant_convolution.hpp> 2 input version: Standard quantized convolution operation inputs = {A_mat, W_mat}

4 input version: Quantized convolution with two sets of scales for A and W matricies. inputs = {A_mat, W_mat, scale_A, scale_W}

Public Functions

inline value **attributes**() const

inline std::string **name**() const

inline void **check_attribute_size**() const

inline *shape* **normalize_compute_shape**(std::vector<*shape*> inputs) const

inline size_t **kdims**() const

inline *argument* **compute**(*shape* output_shape, std::vector<*argument*> args) const

Public Members

std::vector<std::size_t> **padding** = {0, 0} std::vector<std::size_t> **stride** = {1, 1}

std::vector<std::size_t> **dilation** = {1, 1}

padding_mode_t **padding_mode** = default_

int **group** = 1

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **quant_dot**

#include <migraphx/op/quant_dot.hpp> 2 input version: Standard quantized GEMM operation inputs = {A_mat, B_mat}

4 input version: Quantized GEMM with two sets of scales for A and B matricies. inputs = {A_mat, B_mat, scale_A, scale_B}

Public Functions

inline value **attributes**() const

inline std::string **name**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

struct **quantizelinear**

#include <migraphx/op/quantizelinear.hpp>

Public Functions

inline std::string **name**() const

inline value **attributes**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const *shape* &output_shape, std::vector<*argument*> args) const

Public Members

std::optional<migraphx::*shape*::type_t> **out_type**

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **random_seed**

#include <migraphx/op/random_seed.hpp> Generates a random seed for the use of random number generators. Generating the seed at runtime guarantees there will be a different random sequence on every execution. This operation has no inputs or attributes, and outputs an unsigned integer tensor with a single value.

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(const std::vector<*shape*> &inputs) const

inline *argument* **compute**(const *shape* &output_shape, const std::vector<*argument*>&) const

Public Members

shape::type_t **dtype** = *shape*::type_t::uint64_type

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **random_uniform**

#include <migraphx/op/random_uniform.hpp> *random_uniform* populates the passed shape with random numbers, in a uniform distribution. Range for floating-point data types is (0, 1); for integer types it is [0, <max value for the type>]

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const dyn_output &dyn_out, std::vector<*argument*> args) const

inline std::vector<std::size_t> **output_alias**(const std::vector<*shape*>&) const

struct **recip** : public migraphx::internal::op::unary<*recip*>

#include <migraphx/op/recip.hpp>

Public Functions

inline std::string **point_op**() const

inline auto **apply**() const

struct **reduce_all** : public migraphx::internal::op::reduce_op<*reduce_all*>

#include <migraphx/op/reduce_all.hpp>

Public Functions

inline **reduce_all**()

inline **reduce_all**(std::vector<int64_t> ax)

inline auto **op**() const

inline auto **init**() const

struct **reduce_any** : public migraphx::internal::op::reduce_op<*reduce_any*>

#include <migraphx/op/reduce_any.hpp>

Public Functions

inline **reduce_any**()

inline **reduce_any**(std::vector<int64_t> ax)

inline auto **op**() const

struct **reduce_max** : public migraphx::internal::op::reduce_op<*reduce_max*>

#include <migraphx/op/reduce_max.hpp>

Public Functions

inline **reduce_max**()

inline **reduce_max**(std::vector<int64_t> ax)

inline auto **op**() const

inline auto **init**() const

struct **reduce_mean** : public migraphx::internal::op::reduce_op<*reduce_mean*>

#include <migraphx/op/reduce_mean.hpp>

Public Functions

inline **reduce_mean**()

inline **reduce_mean**(std::vector<int64_t> ax)

inline auto **op**() const

inline auto **output**(const *shape* &s) const

struct **reduce_min** : public migraphx::internal::op::reduce_op<*reduce_min*>

#include <migraphx/op/reduce_min.hpp>

Public Functions

inline **reduce_min**()

inline **reduce_min**(std::vector<int64_t> ax)

inline auto **op**() const

inline auto **init**() const

template<class **Derived**>

struct **reduce_op** : public migraphx::internal::op::op_name<*Derived*>

#include <migraphx/op/reduce_op.hpp>

Public Functions

inline value **attributes**() const

inline *shape* **collapse_reduced_axes**(const *shape* &original_shape, const std::vector<int64_t> &reduce_axes) const

inline *shape* **compute_dynamic_shape**(const std::vector<*shape*> &inputs) const

inline *shape* **compute_static_shape**(const std::vector<*shape*> &inputs) const

inline *shape* **normalize_compute_shape**(std::vector<*shape*> inputs) const

returns a shape in which the axis or axes named for reduction by this op are set, to size 1.

Parameters

inputs – list of input shapes

Returns

shape

template<class **T**>
inline void **tune_dims**(const std::vector<int64_t> &tuned_axes, const std::vector<*T*> &in_lens, std::vector<*T*> &out_lens) const

template<class **T**>
inline void **reduce**(const tensor_view<*T*> &input, const *shape* &batch_shape, const std::vector<int64_t> &tuned_axes, const std::vector<std::size_t> &out_idx, tensor_view<*T*> &output) const

inline *argument* **reduce**(const *shape* &computed_shape, const std::vector<int64_t> &reduce_axes, *argument* &data_arg) const

inline *argument* **compute**(const dyn_output &dyn_out, std::vector<*argument*> args) const

inline auto **init**() const

inline auto **input**() const

inline auto **output**(const *shape*&) const

inline **reduce_op**()

inline **reduce_op**(std::vector<int64_t> ax)

Public Members

std::vector<std::int64_t> **axes** = { }

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **reduce_prod** : public migraphx::internal::op::reduce_op<*reduce_prod*>

#include <migraphx/op/reduce_prod.hpp>

Public Functions

inline **reduce_prod**()

inline **reduce_prod**(std::vector<int64_t> ax)

inline auto **op**() const

inline auto **init**() const

struct **reduce_sum** : public migraphx::internal::op::reduce_op<*reduce_sum*>

#include <migraphx/op/reduce_sum.hpp>

Public Functions

inline **reduce_sum**()

inline **reduce_sum**(std::vector<int64_t> ax)

inline auto **op**() const

struct **relu** : public migraphx::internal::op::unary<*relu*>

#include <migraphx/op/relu.hpp>

Public Functions

inline std::string **point_op**() const

inline auto **apply**() const

struct **reshape**

#include <migraphx/op/reshape.hpp> 1 input version: reshape(input_data) this.dims = output_dims Makes a copy of input_data to the output shape.

2 input version: reshape(input_data, output_buffer) this.dims = unset Copies input_data to output_buffer; output_buffer already has the output shape. This version will not fail gracefully if the input shape and output_buffer shape are incompatible. There's a throw that will catch when the number of elements do not match at runtime. This version should only be used for dynamic reshapes (output dimensions only known at runtime). If output_buffer has a static shape during compile/parse, you can use the 1 input version.

Public Functions

inline std::string **name**() const

inline *shape* **dyn_1arg_compute_shape**(*shape* s0) const

inline *shape* **static_compute_shape**(std::vector<*shape*> inputs, std::size_t n_neg_dims) const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const dyn_output &dyn_out, std::vector<*argument*> args) const

Public Members

std::vector<int64_t> **dims**

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **reshape_lazy**

#include <migraphx/op/reshape_lazy.hpp>

Public Functions

inline value **attributes**() const

inline std::string **name**() const

inline *shape* **dyn_compute_shape**(*shape* s0) const

inline *shape* **static_compute_shape**(std::vector<*shape*> inputs, std::size_t n_neg_dims) const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const dyn_output &dyn_out, std::vector<*argument*> args) const

inline std::vector<std::size_t> **output_alias**(const std::vector<*shape*>&) const

Public Members

std::vector<int64_t> **dims**

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

template<class **Iterator**>
static inline auto **compute_end_dim**(*Iterator* start, *Iterator* last, std::size_t dim)

template<class **OptionalPair**>
static inline *OptionalPair* **try_merge_pairs**(*OptionalPair* p2, *OptionalPair* p1)

template<class **DimIterator**, class **StrideIterator**>

static inline optional<std::size_t> **merge_strides**(*DimIterator* dim_start, *DimIterator* dim_last, *StrideIterator* stride_start, *StrideIterator* stride_last)

template<class **DimIterator**, class **StrideIterator**>
static inline auto **can_strides_merge**(*DimIterator* dim_start, *DimIterator* dim_last, *StrideIterator* stride_start, *StrideIterator* stride_last)

static inline optional<*shape*> **reshape_lazy_dims**(const *shape* &input, const std::vector<std::size_t> &rdims)

struct **resize**

#include <migraphx/op/resize.hpp> The Resize operation mirrors the Onnx Resize operation with some differences. Currently, only Nearest mode is supported. "Axes" and "ROI" attributes not recognized.

Accepts either one or two runtime inputs. Input 0 - data to be resized Input 1 - sizes or scales. If data type is uint64, Input 1 is interpreted as output sizes; otherwise as scaling factors.

If the second input is not used, either a "sizes" or "scales" attribute must be provided.

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const migraphx::*shape*&, std::vector<*argument*> args) const

Public Members

std::vector<float> **scales**

std::vector<size_t> **sizes**

std::string **nearest_mode** = {"floor"}

std::string **mode** = {"nearest"}

std::string **coordinate_transformation_mode**

Public Static Functions

static inline auto &**get_nearest_op**(const std::string &near_mode)

static inline auto &**get_original_idx_op**(const std::string &s_mode)

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **reverse**

#include <migraphx/op/reverse.hpp>

Public Functions

inline std::string **name**() const

inline value **attributes**() const

inline *shape* **normalize_compute_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const *shape* &s, std::vector<*argument*> args) const

Public Members

std::vector<int64_t> **axes**

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **rnn**

#include <migraphx/op/rnn.hpp>

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

Public Members

std::size_t **hidden_size** = 1

std::vector<operation> **actv_funcs** = {*tanh*{}, *tanh*{}}

rnn_direction **direction** = *rnn_direction*::*forward*

float **clip** = 0.0f

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **rnn_last_cell_output**

#include <migraphx/op/rnn_last_cell_output.hpp>

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const struct **rnn_last_hs_output**

#include <migraphx/op/rnn_last_hs_output.hpp>

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

struct **rnn_var_sl_last_output**

#include <migraphx/op/rnn_var_sl_last_output.hpp>

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

Public Members

rnn_direction **direction** = *rnn_direction*::*forward*

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **rnn_var_sl_shift_output**

#include <migraphx/op/rnn_variable_seq_lens.hpp>

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const *shape* &output_shape, std::vector<*argument*> args) const

Public Members

std::string **output_name** = "hidden_states"

rnn_direction **direction** = *rnn_direction*::*forward*

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **rnn_var_sl_shift_sequence**

#include <migraphx/op/rnn_variable_seq_lens.hpp>

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const *shape* &output_shape, std::vector<*argument*> args) const

struct **roialign**

#include <migraphx/op/roialign.hpp>

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

inline auto **calc_pos_weight**(const std::array<std::size_t, 2> &dims, const *shape* &comp_s, const std::array<float, 2> &roi_start, const std::array<float, 2> &bin_size, const std::array<std::size_t, 2> &bin_grid_size) const

template<class **T**, class **Op**>
inline std::tuple<double, int64_t> **calc_pooling**(const *T* &data, const std::array<std::size_t, 2> &bin_grid_size, const std::vector<*pos_weight*> &pos_weights, int64_t index, *Op* op) const

inline *argument* **compute**(const *shape* &output_shape, std::vector<*argument*> args) const

Public Members

std::string **coord_trans_mode** = "half_pixel"

pooling_mode **mode** = {*pooling_mode*::*average*}

int64_t **output_height** = 1

int64_t **output_width** = 1

int64_t **sampling_ratio** = 0

float **spatial_scale** = 1.0f

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **avg_pool**

#include <migraphx/op/roialign.hpp>

Public Functions

inline double **init**()

inline double **operator()**(double x, double y)

inline double **final**(double x, std::size_t y)

struct **max_pool**

#include <migraphx/op/roialign.hpp>

Public Functions

inline double **init**()

inline double **operator()**(double x, double y)

inline double **final**(double x, std::size_t)

struct **pos_weight**

#include <migraphx/op/roialign.hpp>

Public Members

std::array<std::size_t, 4> **pos** = {0, 0, 0, 0}

std::array<float, 4> **w** = {0.0f, 0.0f, 0.0f, 0.0f}

struct **rsqrt** : public migraphx::internal::op::unary<*rsqrt*>

#include <migraphx/op/rsqrt.hpp>

Public Functions

inline auto **apply**() const

struct **run_on_target**

#include <migraphx/op/run_on_target.hpp>

Public Functions

inline std::string **name**() const

inline migraphx::*shape* **compute_shape**(const std::vector<migraphx::*shape*> &inputs, std::vector<migraphx::module_ref> mods) const

inline migraphx::*argument* **compute**(const migraphx::*shape*&, const std::vector<migraphx::*argument*> &args, const std::vector<migraphx::module_ref> &mods, const std::function<std::vector<migraphx::*argument*>(migraphx::module_ref&, const std::unordered_map<std::string, migraphx::*argument*>&)> &run) const

Public Members

std::size_t **target_id** = 0

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **scalar**

#include <migraphx/op/scalar.hpp>

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(*shape* output_shape, std::vector<*argument*> args) const

inline std::vector<std::size_t> **output_alias**(const std::vector<*shape*>&) const

Public Members

std::vector<std::size_t> **scalar_bcast_lens**

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **scan_slice** : public migraphx::internal::op::op_name<*scan_slice*>

#include <migraphx/op/scan_slice.hpp>

Public Functions

inline value **attributes**() const

inline *shape* **normalize_compute_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(*shape* output_shape, std::vector<*argument*> args) const

Public Members

int64_t **axis** = 0

int64_t **direction** = 0

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **scatter_add** : public migraphx::internal::op::scatter_op<*scatter_add*>

#include <migraphx/op/scatter_add.hpp>

Public Functions

inline auto **reduction**() const

struct **scatter_max** : public migraphx::internal::op::scatter_op<*scatter_max*>

#include <migraphx/op/scatter_max.hpp>

Public Functions

inline auto **reduction**() const

struct **scatter_min** : public migraphx::internal::op::scatter_op<*scatter_min*>

#include <migraphx/op/scatter_min.hpp>

Public Functions

inline auto **reduction**() const

struct **scatter_mul** : public migraphx::internal::op::scatter_op<*scatter_mul*>

#include <migraphx/op/scatter_mul.hpp>

Public Functions

inline auto **reduction**() const

struct **scatter_none** : public migraphx::internal::op::scatter_op<*scatter_none*>

#include <migraphx/op/scatter_none.hpp>

Public Functions

inline auto **reduction**() const

template<typename **Derived**>

struct **scatter_op** : public migraphx::internal::op::op_name<*Derived*>

#include <migraphx/op/scatter_op.hpp>

Public Functions

inline value **attributes**() const

inline *shape* **normalize_compute_shape**(std::vector<*shape*> inputs) const

template<class **TensorView0**, class **TensorView1**>

inline void **scatter_reduce_iterate**(*TensorView0* indices, *TensorView1* output, *TensorView1* update) const

inline *argument* **compute**(const *shape* &output_shape, std::vector<*argument*> args) const

inline const *Derived* &**derived**() const

Public Members

int64_t **axis** = 0

bool **skip_out_of_bounds** = false

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **scatternd_add** : public migraphx::internal::op::scatternd_op<*scatternd_add*>

#include <migraphx/op/scatternd_add.hpp>

Public Functions

inline **scatternd_add**()

inline auto **reduction**() const

struct **scatternd_max** : public migraphx::internal::op::scatternd_op<*scatternd_max*>

#include <migraphx/op/scatternd_max.hpp>

Public Functions

inline **scatternd_max**()

inline auto **reduction**() const

struct **scatternd_min** : public migraphx::internal::op::scatternd_op<*scatternd_min*>

#include <migraphx/op/scatternd_min.hpp>

Public Functions

inline **scatternd_min**()

inline auto **reduction**() const

struct **scatternd_mul** : public migraphx::internal::op::scatternd_op<*scatternd_mul*>

#include <migraphx/op/scatternd_mul.hpp>

Public Functions

inline **scatternd_mul**()

inline auto **reduction**() const

struct **scatternd_none** : public migraphx::internal::op::scatternd_op<*scatternd_none*>

#include <migraphx/op/scatternd_none.hpp>

Public Functions

inline **scatternd_none**()

inline auto **reduction**() const

template<class **Derived**>

struct **scatternd_op** : public migraphx::internal::op::op_name<*Derived*>

#include <migraphx/op/scatternd_op.hpp> N-dimensional Scatter operations. This struct is parent class to ops which differ in what formula is used to reduce (combine old and new values of) the scattered value. It was originally based on Onnx ScatterND operation (see https://github.com/onnx/onnx/blob/main/docs/Operators.md#ScatterND) and is also similar to Numpy numpy.add.at().

Template Parameters

Derived – a template parameter in the CRTP inheritance idiom, represents one of the child operations.

Public Functions

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

Validate input shapes and return the correct output shape. For Scatter ops, the output is the same shape as the data tensor (first input), but cast to a standard shape.

inline *argument* **compute**(const dyn_output &dyn_out, std::vector<*argument*> args) const

inline auto **init**() const

inline **scatternd_op**()

struct **select_module**

#include <migraphx/op/select_module.hpp>

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(const std::vector<*shape*> &inputs, const std::vector<module_ref>&) const

inline std::vector<std::string> **get_input_parameter_names**(module_ref mod) const

inline std::vector<std::string> **get_output_parameter_names**(module_ref mod) const

inline *argument* **compute**(const *shape*&, const std::vector<*argument*> &args, const std::vector<module_ref> &submodule_list, const std::function<std::vector<*argument*>(module_ref&, const std::unordered_map<std::string, *argument*>&)> &run) const

inline std::vector<std::size_t> **output_alias**(const std::vector<*shape*> &shapes) const

Public Members

shape **output_dyn_shapes**

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **sigmoid** : public migraphx::internal::op::unary<*sigmoid*>

#include <migraphx/op/sigmoid.hpp>

Public Functions

inline std::string **point_op**() const

inline auto **apply**() const

struct **sign** : public migraphx::internal::op::unary<*sign*>

#include <migraphx/op/sign.hpp>

Public Functions

inline std::string **point_op**() const

inline auto **apply**() const

struct **sin** : public migraphx::internal::op::unary<*sin*>

#include <migraphx/op/sin.hpp>

Public Functions

inline auto **apply**() const

struct **sinh** : public migraphx::internal::op::unary<*sinh*>

#include <migraphx/op/sinh.hpp>

Public Functions

inline auto **apply**() const

struct **slice**

#include <migraphx/op/slice.hpp> Slice operator that accepts variable axes, starts and ends. All of `starts`, `ends`, and `axes` must be supplied by either their attribute or an input (but not both).

Valid calls: slice(input); axes, starts, ends set slice(input, starts); axes, ends set slice(input, ends); starts, axes set slice(input, axes); starts, ends set slice(input, starts, ends); axes set slice(input, starts, axes); ends set slice(input, ends, axes); starts set slice(input, start, ends, axes); none set

Attributes: axes: constant axes to slice over (optional) starts: constant slice starting indices (optional) ends: constant slice ending indices (optional)

Parameters: data: the input tensor to slice (dynamic or static shape) input_starts: starting indices of slice (optional, static shape) input_ends: ending indices of slice (optional, static shape) input_axes: axes to slice over (optional, static shape)

Public Functions

inline value **attributes**() const

Ensure that attribute axes is within limits. Will attempt to normalize starts and ends; but will use the dynamic_dimension.max values for dynamic shapes. This makes it so you have to renormalize for non-fixed dynamic_dimensions.

inline std::string **name**() const

template<class **A**, class **B**>
inline std::vector<std::size_t> **lens_calc**(const std::vector<std::size_t> &lengths, *A* in_starts, *A* in_ends, *B* in_axes) const

Computes the slice output shape dimensions for given starts, ends,and axes. Templated to also handle tensor views. Possibly different type between [in_starts, in_ends] and [in_axes] if in_axes is this object's axes attribute. Assumes in_starts and in_ends are normalized; in_axes are valid.

inline std::array<bool, 3> **get_set_attributes**() const

Get the attributes that are non-empty.

inline *shape* **compute_two_or_more**(std::vector<*shape*> inputs) const

Helper function for *normalize_compute_shape()*

inline *shape* **normalize_compute_shape**(std::vector<*shape*> inputs) const

inline auto **compute_offset**(const *shape* &s) const

Calculates the starting offset for the sliced tensor. Used in compute when only data input and all other information are in the attributes.

Parameters

s – static input shape

template<class **T**>
inline auto **compute_offset**(const *shape* &s, const *T* &input_starts, const *T* &ax_vec) const

Calculates the starting offset for the sliced tensor (for aliasing). Used for 2-4 inputs to `slice.

Parameters

- **s** – static input shape
- **input_starts** – starting indices of slice
- **ax_vec** – axes to slice on

inline std::unordered_map<std::string, std::vector<int64_t>> **normalize_starts_ends_axes**(*shape* input_shape, const optional<std::vector<int64_t>> &input_starts, const optional<std::vector<int64_t>> &input_ends, const optional<std::vector<int64_t>> &input_axes) const

If given, normalize the inputs. Otherwise get from operator attributes. Return the values in a map.

Parameters input_shape: static shape of the input input_starts: optional input_ends: optional input_ends: optional

inline *argument* **compute**(const dyn_output &dyn_out, std::vector<*argument*> args) const

inline std::vector<std::size_t> **output_alias**(const std::vector<*shape*>&) const

Public Members

std::vector<int64_t> **axes** = {}

std::vector<int64_t> **starts** = {}

std::vector<int64_t> **ends** = {}

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

Public Static Attributes

static constexpr std::array<bool, 3> **all_set** = {true, true, true}

Named arrays for the set attribute possibilities.

static constexpr std::array<bool, 3> **ends_axes** = {false, true, true}

static constexpr std::array<bool, 3> **starts_axes** = {true, false, true} static constexpr std::array<bool, 3> **starts_ends** = {true, true, false}

static constexpr std::array<bool, 3> **axes_only** = {false, false, true}

static constexpr std::array<bool, 3> **ends_only** = {false, true, false}

static constexpr std::array<bool, 3> **starts_only** = {true, false, false}

static constexpr std::array<bool, 3> **none_set** = {false, false, false}

struct **softmax**

#include <migraphx/op/softmax.hpp>

Public Functions

inline value **attributes**() const

inline std::string **name**() const

inline *shape* **normalize_compute_shape**(std::vector<*shape*> inputs) const

inline auto **output**() const

Public Members

int64_t **axis** = 1

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **sqdiff** : public migraphx::internal::op::binary<*sqdiff*>

#include <migraphx/op/sqdiff.hpp>

Public Functions

inline std::string **point_op**() const

inline auto **apply**() const

struct **sqrt** : public migraphx::internal::op::unary<*sqrt*>

#include <migraphx/op/sqrt.hpp>

Public Functions

inline auto **apply**() const

struct **squeeze**

#include <migraphx/op/squeeze.hpp>

Public Functions

inline value **attributes**() const

inline std::string **name**() const

inline *shape* **normalize_compute_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const dyn_output &dyn_out, std::vector<*argument*> args) const

inline std::vector<std::size_t> **output_alias**(const std::vector<*shape*>&) const

Public Members

std::vector<int64_t> **axes**

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **step**

#include <migraphx/op/step.hpp>

Public Functions

inline value **attributes**() const

inline std::string **name**() const

inline *shape* **normalize_compute_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(*shape* output_shape, std::vector<*argument*> args) const

inline std::vector<std::size_t> **output_alias**(const std::vector<*shape*>&) const

Public Members

std::vector<int64_t> **axes**

std::vector<int64_t> **steps**

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **sub** : public migraphx::internal::op::binary<*sub*>

#include <migraphx/op/sub.hpp>

Public Functions

inline std::string **point_function**() const

inline auto **apply**() const

struct **tan** : public migraphx::internal::op::unary<*tan*>

#include <migraphx/op/tan.hpp>

Public Functions

inline auto **apply**() const

struct **tanh** : public migraphx::internal::op::unary<*tanh*>

#include <migraphx/op/tanh.hpp>

Public Functions

inline auto **apply**() const

struct **topk**

#include <migraphx/op/topk.hpp>

Public Functions

inline value **attributes**() const

inline std::string **name**() const

inline *shape* **normalize_compute_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const *shape* &output_shape, std::vector<*argument*> args) const

Public Members

int64_t **k** = 1

int64_t **axis** = 0

bool **largest** = true

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

template<class **Compare**>
static inline auto **compare_pair**(*Compare* compare)

struct **transpose**

#include <migraphx/op/transpose.hpp>

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const dyn_output &dyn_out, std::vector<*argument*> args) const

inline std::vector<std::size_t> **output_alias**(const std::vector<*shape*>&) const

Public Members

std::vector<int64_t> **dims**

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

template<class **Derived**>

struct **unary** : public migraphx::internal::op::op_name<*Derived*>

#include <migraphx/op/unary.hpp>

Public Functions

inline std::string **point_function**() const

inline std::string **point_op**() const

inline value **base_attributes**() const

inline value **attributes**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const dyn_output &dyn_out, std::vector<*argument*> args) const

struct **unary_not** : public migraphx::internal::op::unary<*unary_not*>

#include <migraphx/op/unary_not.hpp>

Public Functions

inline std::string **point_function**() const

inline auto **apply**() const

inline std::string **name**() const

struct **undefined**

#include <migraphx/op/undefined.hpp>

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(const std::vector<*shape*> &inputs) const

inline *argument* **compute**(const *shape*&, const std::vector<*argument*>&) const

struct **unique**

#include <migraphx/op/unique.hpp>

Public Functions

template<class **T**>
inline auto **make_idx_less_fn**(const *T* &data, size_t chunk_sz) const

template<class **T**>
inline auto **sorted_uniq_indices**(const *T* &input_data, size_t chunk_sz) const

template<class **T**>
inline auto **unsorted_uniq_indices**(const *T* &input_data, size_t chunk_sz) const

inline std::string **name**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const dyn_output &dyn_out, std::vector<*argument*> args) const

Public Members

std::optional<int64_t> **axis**

bool **sorted** = true

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **unknown**

#include <migraphx/op/unknown.hpp>

Public Functions

inline std::string **name**() const

inline *shape* **compute_shape**(std::vector<*shape*> input) const

Public Members

std::string **op**

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

Friends

inline friend std::ostream &**operator<<**(std::ostream &os, const *unknown* &x)

struct **unpack_fp4**

#include <migraphx/op/unpack_fp4.hpp>

Public Functions

inline std::string **name**() const

inline migraphx::*shape* **normalize_compute_shape**(std::vector<migraphx::*shape*> inputs) const

inline *argument* **compute**(const *shape* &output_shape, const std::vector<*argument*> &args) const

struct **unpack_int4**

#include <migraphx/op/unpack_int4.hpp>

Public Functions

inline std::string **name**() const

inline value **attributes**() const

inline migraphx::*shape* **normalize_compute_shape**(std::vector<migraphx::*shape*> inputs) const

inline *argument* **compute**(const *shape* &output_shape, std::vector<*argument*> args) const

Public Members

int64_t **axis** = -1

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **unsqueeze**

#include <migraphx/op/unsqueeze.hpp> Adds dimensions to a tensor based on the axes attribute. `axes` are based on the number of output shape dimensions and should not contain duplicates. `steps` are for modifying dimensions added to the middle of the original shape. Each step must be a factor of the original dimension. ex: unsqueeze(shape = [3, 4, 10], axes = [2, 4, 5], steps = [2]) -> shape = [3, 4, 2, 5, 1, 1] Dynamic shape version does not handle `steps`.

Public Functions

inline value **attributes**() const

inline std::string **name**() const

inline *shape* **normalize_compute_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(const dyn_output &dyn_out, std::vector<*argument*> args) const

inline std::vector<std::size_t> **output_alias**(const std::vector<*shape*>&) const

Public Members

std::vector<int64_t> **axes**

std::vector<int64_t> **steps**

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **where**

#include <migraphx/op/where.hpp>

Public Functions

inline std::string **name**() const

inline value **attributes**() const

inline *shape* **compute_shape**(std::vector<*shape*> inputs) const

inline *argument* **compute**(*shape* output_shape, std::vector<*argument*> args) const

struct **zero**

#include <migraphx/op/reduce_op.hpp>

Public Functions

template<class **T**>
inline **operator** *T*() const

namespace **builder**

Typedefs

using **builder_func** = std::function<std::vector<instruction_ref>(module &m, instruction_ref ins, const std::vector<instruction_ref> &args, const std::vector<module_ref> &module_args, const value &options)>

Functions

std::vector<instruction_ref> **insert**(const std::string &name, module &m, instruction_ref ins, const std::vector<instruction_ref> &args, const value &options)

std::vector<instruction_ref> **insert**(const std::string &name, module &m, instruction_ref ins, const std::vector<instruction_ref> &args, const std::vector<module_ref> &module_args, const value &options)

std::vector<instruction_ref> **add**(const std::string &name, module &m, const std::vector<instruction_ref> &args, const value &options)

std::vector<instruction_ref> **add**(const std::string &name, module &m, const std::vector<instruction_ref> &args, const std::vector<module_ref> &module_args, const value &options)

template<class ...**Ins**>
instruction_ref **insert_common_op**(module &m, instruction_ref ins, const std::string &op_name, *Ins*... args)

void **register_builder**(const std::string &name, *builder_func* f)

template<class **T**>
auto **invoke_builder**(module &m, instruction_ref ins, const std::vector<instruction_ref> &args, const std::vector<module_ref> &module_args, const value &options) -> decltype(*T*{}.insert(*m*, *ins*, *args*, *module_args*))

template<class **T**>
void **register_builder**()

static std::unordered_map<std::string, *builder_func*> &**builder_map**()

struct **batchnorm** : public migraphx::internal::op::builder::op_builder<*batchnorm*>

Public Functions

inline std::vector<instruction_ref> **insert**(module &m, instruction_ref ins, const std::vector<instruction_ref> &args) const

Public Members

float **epsilon** = 1e-5f

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **celu** : public migraphx::internal::op::builder::op_builder<*celu*>

Public Functions

inline std::vector<instruction_ref> **insert**(module &m, instruction_ref ins, const std::vector<instruction_ref> &args) const

Public Members

float **alpha** = 1.0f

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **clip** : public migraphx::internal::op::builder::op_builder<*clip*>

Public Functions

inline std::vector<instruction_ref> **insert**(module &m, instruction_ref ins, const std::vector<instruction_ref> &args) const

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self*&, *F*)

struct **convolution** : public migraphx::internal::op::builder::convolution_base<*convolution*>

Public Functions

inline std::vector<instruction_ref> **insert**(module &m, instruction_ref ins, const std::vector<instruction_ref> &args)

inline instruction_ref **add_bias**(module &m, instruction_ref ins, const std::vector<instruction_ref> &args, instruction_ref curr_ins, uint64_t axis) const

Public Static Functions

static inline std::string **name**()

template<class **Derived**>

struct **convolution_base** : public migraphx::internal::op::builder::op_builder<*Derived*>

Public Functions

inline void **validate_or_init_attributes**(size_t kdims, const instruction_ref x, const instruction_ref w)

inline bool **is_dynamic**(const *shape* &s) const

inline operation **make_conv_op**(const std::string &name) const

Public Members

std::string **auto_pad** = "NOTSET"

std::vector<int64_t> **paddings**

std::vector<std::size_t> **strides**

std::vector<std::size_t> **dilations**

int **group** = 1

padding_mode_t **padding_mode** = *padding_mode_t*::*default_*

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **einsum** : public migraphx::internal::op::builder::op_builder<*einsum*>

Public Types

using **int_mat** = std::vector<std::vector<int>>

Public Functions

inline std::vector<instruction_ref> **insert**(module &m, instruction_ref ins, const std::vector<instruction_ref> &args) const

inline *equation_info* **analyze_equation**(const std::vector<instruction_ref> &args) const

inline *equation_info* **parse_equation**() const

inline size_t **validate_input_terms**(const std::vector<std::string> &input_terms, const std::vector<instruction_ref> &args) const

inline void **validate_output_term**(std::string_view output_term, const std::map<char, int> &label_count, size_t ellipsis_ndim) const

inline std::string **generate_output_term**(const std::map<char, int> &label_count, size_t ellipsis_ndim) const

inline *int_mat* **make_mapping_matrix**(const std::vector<std::string> &terms, const std::map<char, int> &label_count, size_t ellipsis_ndim) const

inline std::vector<std::map<char, std::vector<int>>> **find_duplicates**(const std::vector<std::string> &terms) const inline instruction_ref **preprocess_input**(module &m, instruction_ref ins, instruction_ref op, const std::map<char, std::vector<int>> &duplicates, const *int_mat* &map_mat, size_t input_idx, *int_mat* &cur_pair) const

inline instruction_ref **gather_diagonal**(module &m, instruction_ref ins, *int_mat* &cur_pair, instruction_ref op, const *int_mat* &diag) const

inline instruction_ref **process_pair**(module &m, instruction_ref ins, instruction_ref op1, instruction_ref op2, const *int_mat* &map_mat, size_t input_idx, *int_mat* &cur_pair) const

inline instruction_ref **batch_dot**(module &m, instruction_ref ins, *int_mat* &cur_pair, instruction_ref op1, instruction_ref op2, const std::vector<int> &batch_axes, const std::vector<int> &sum_axes) const

inline instruction_ref **finalize_output**(module &m, instruction_ref ins, instruction_ref op, const *int_mat* &map_mat, *int_mat* &cur_pair) const

inline instruction_ref **transpose_unsqueeze**(module &m, instruction_ref ins, *int_mat* &cur_pair, instruction_ref op) const

inline instruction_ref **squeeze_transpose**(module &m, instruction_ref ins, *int_mat* &cur_pair, instruction_ref op, std::vector<int> row_output) const

inline instruction_ref **apply_transpose_op**(module &m, instruction_ref ins, instruction_ref op, const std::vector<int64_t> &perm, std::vector<int> &row) const

inline std::pair<instruction_ref, instruction_ref> **apply_broadcast_op**(module &m, instruction_ref ins, instruction_ref opl, instruction_ref opr, const std::vector<int> &common_labels) const

inline instruction_ref **apply_reduce_sum_op**(module &m, instruction_ref ins, instruction_ref op, const std::vector<int> &axes, std::vector<int> &row) const

inline *int_mat* **make_matrix**(int cur_pair, int cols, int fill_value) const

inline std::vector<int> **extract_column**(*int_mat* map_mat, int col_idx, int row_begin, int row_end) const

inline std::vector<int> **set_union**(const std::vector<int> &lhs, const std::vector<int> &rhs) const

inline std::vector<int> **set_difference**(const std::vector<int> &lhs, const std::vector<int> &rhs) const

inline std::vector<int> **arange**(int start_value, int end_value) const

template<class **Vec**, class ...**Vecs**>
inline *Vec* **concat_vectors**(*Vec* vec, *Vecs*&&... vecs) const

inline size_t **calc_dim**(const std::vector<int> &axes, const std::vector<size_t> &lens) const

Public Members

std::string **equation** = ""

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **equation_info**

Public Members

bool **explicit_form** = false

std::vector<std::string> **input_terms**

std::string **output_term**

std::map<char, int> **label_count**

std::vector<std::map<char, std::vector<int>>> **duplicates**

size_t **ellipsis_ndim** = 0

struct **gelu_erf** : public migraphx::internal::op::builder::op_builder<*gelu_erf*>

Public Functions

inline std::vector<instruction_ref> **insert**(module &m, instruction_ref ins, const std::vector<instruction_ref> &args) const

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self*&, *F*)

struct **gelu_quick** : public migraphx::internal::op::builder::op_builder<*gelu_quick*>

Public Functions

inline std::vector<instruction_ref> **insert**(module &m, instruction_ref ins, const std::vector<instruction_ref> &args) const

Public Members

float **alpha** = 1.0f

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **gelu_split** : public migraphx::internal::op::builder::op_builder<*gelu_split*>

Public Functions

inline std::vector<instruction_ref> **insert**(module &m, instruction_ref ins, const std::vector<instruction_ref> &args) const

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self*&, *F*)

struct **gelu_tanh** : public migraphx::internal::op::builder::op_builder<*gelu_tanh*>

Public Functions

inline std::vector<instruction_ref> **insert**(module &m, instruction_ref ins, const std::vector<instruction_ref> &args) const

Public Members

bool **fast** = false

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **gemm** : public migraphx::internal::op::builder::op_builder<*gemm*>

Public Functions

inline std::vector<instruction_ref> **insert**(module &m, instruction_ref ins, const std::vector<instruction_ref> &args) const

Public Members

float **alpha** = 1.0f

float **beta** = 1.0f

bool **trans_a** = false

bool **trans_b** = false

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

struct **mean_variance_normalization** : public migraphx::internal::op::builder::op_builder<*mean_variance_normalization*>

Public Functions

inline std::vector<instruction_ref> **insert**(module &m, instruction_ref ins, const std::vector<instruction_ref> &args) const

Public Members

std::vector<int64_t> **axes** = {0, 2, 3}

Public Static Functions

template<class **Self**, class **F**>
static inline auto **reflect**(*Self* &self, *F* f)

template<class **T**>

struct **op_builder** : public migraphx::internal::auto_register<*register_builder_action*, *T*>

#include </home/docs/checkouts/readthedocs.org/user_builds/advanced-micro-devices-amdmigraphx/checkouts/latest/src/op/builder/include/migraphx/op/builder/op_builder.hpp>
Subclassed by *migraphx::internal::op::builder::convolution_base< convolution >*, *migraphx::internal::op::builder::convolution_base< quant_convolution >*

Public Static Functions

static inline std::string **name**()

struct **quant_convolution** : public migraphx::internal::op::builder::convolution_base<*quant_convolution*>

Public Functions

inline std::vector<instruction_ref> **insert**(module &m, instruction_ref ins, const std::vector<instruction_ref> &args)

inline instruction_ref **add_int8_shift**(module &m, instruction_ref ins, const instruction_ref &offset_op, instruction_ref &unshifted_input) const

inline void **shift_input_and_bias**(module &m, instruction_ref ins, const instruction_ref &offset_op, const bool has_bias, instruction_ref &input, instruction_ref &input_bias) const

inline float **get_symmetric_value**(const instruction_ref &input) const

inline instruction_ref **gen_symmetric_literal**(module &m, const instruction_ref &input) const inline instruction_ref **get_zero_point**(module &m, const instruction_ref &input, int index, const std::vector<instruction_ref> &args) const

inline bool **is_symmetric_zero_point**(instruction_ref zp) const

inline auto **qparam_broadcast_op**(instruction_ref qparam, std::vector<std::size_t> lens, std::size_t axis) const

inline instruction_ref **handle_quant_bias**(module &m, instruction_ref ins, const operation &op, const instruction_ref &input, const instruction_ref &x, const instruction_ref &weights, const instruction_ref &x_zp, const instruction_ref &w_zp) const

inline void **handle_quant_inputs**(module &m, instruction_ref ins, instruction_ref &input, instruction_ref &weights, instruction_ref &input_zp, instruction_ref &weight_zp) const

Public Static Functions

static inline std::string **name**()

struct **register_builder_action**

#include </home/docs/checkouts/readthedocs.org/user_builds/advanced-micro-devices-amdmigraphx/checkouts/latest/src/op/builder/include/migraphx/op/builder/op_builder.hpp>

Public Static Functions

template<class **T**>
static inline void **apply**()

5.6 Program

5.6.1 instruction

struct **instruction**

Public Functions

inline **instruction**()

instruction(*operation* o, *shape* r, std::vector<*instruction_ref*> args)

instruction(*operation* o, *shape* r, std::vector<*instruction_ref*> args, std::vector<module_ref> modules)

instruction(*literal* l)

void **replace**(*operation* o)

void **recompute_shape**()

void **clear_arguments**()

bool **valid**(*instruction_ref* start, bool check_order = false) const bool **valid**() const

const *shape* &**get_shape**() const

const *literal* &**get_literal**() const

const *operation* &**get_operator**() const

std::string **name**() const

const std::vector<*instruction_ref*> &**inputs**() const

const std::vector<module_ref> &**module_inputs**() const

const std::vector<*instruction_ref*> &**outputs**() const

Where this instruction is used as an input to another instruction.

void **add_output**(*instruction_ref* ins)

template<class **T**>
inline void **remove_output**(const *T* &ins)

bool **can_eval**() const

bool **is_undefined**() const

argument **eval**(bool check_eval = true) const

void **finalize**(context &ctx)

void **set_normalized**(bool value = true)

bool **is_normalized**() const

bool **need_normalization**() const

operation **normalized_operator**() const

std::size_t **get_target_id**() const

void **set_target_id**(std::size_t tid)

void **debug_print**() const

Public Static Functions

static void **replace_refs**(*instruction_ref* ins, const std::unordered_map<*instruction_ref*, *instruction_ref*> &map_insts, const std::unordered_map<module_ref, module_ref> &map_mods)

static void **backreference**(*instruction_ref* ref)

static void **replace_argument**(*instruction_ref* ins, *instruction_ref* old, *instruction_ref* new_ins)

static void **replace_mod_argument**(*instruction_ref* ins, module_ref old, module_ref new_mod)

static void **replace**(*instruction_ref* ins, *operation* o, const *shape* &r, std::vector<*instruction_ref*> args)

static void **replace**(*instruction_ref* ins, *operation* o, const *shape* &r, std::vector<*instruction_ref*> args, std::vector<module_ref> module_args) static std::vector<*instruction_ref*> **get_output_alias**(*instruction_ref* ins, bool shallow = false)

static void **print**(std::ostream &os, *instruction_ref* ins, const std::unordered_map<*instruction_ref*, std::string> &names)

Friends

friend bool **operator==**(const *instruction* &i, *instruction_ref* ref)

friend bool **operator==**(const *instruction* &x, const *instruction* &y)

friend bool **operator!=**(const *instruction* &x, const *instruction* &y)

friend bool **operator==**(*instruction_ref* ref, const *instruction* &i)

friend bool **operator!=**(const *instruction* &i, *instruction_ref* ref)

friend bool **operator!=**(*instruction_ref* ref, const *instruction* &i)

5.6.2 instruction_ref

type migraphx::internal::**instruction_ref**

References an instruction in the program.

5.6.3 program

struct **program**

Stores the instruction stream.

Public Functions

program()

explicit **program**(module m)

program(*program*&&) noexcept

program(const *program*&)

program &**operator=**(*program*)

~program() noexcept

std::vector<std::string> **get_parameter_names**() const

shape **get_parameter_shape**(std::string name) const

instruction_ref **get_parameter**(std::string name) const

std::unordered_map<std::string, *shape*> **get_parameter_shapes**() const

std::size_t **total_instructions**() const

std::vector<*argument*> **eval**(const parameter_map &params, execution_environment exec_env = execution_environment{ }) const std::vector<*argument*> **eval_with_context**(std::vector<context> &ctx, const parameter_map &params) const

void **finish**() const

std::size_t **size**() const

std::vector<*shape*> **get_output_shapes**() const

context &**get_context**() const

instruction_ref **validate**() const

target_assignments **get_target_assignments**(const std::vector<*target*> &targets, assignment_options options = assignment_options{ })

void **compile**(const *target* &t, compile_options options = compile_options{ })

void **compile**(const std::vector<*target*> &targets, std::vector<compile_options> compile_opts = { })

bool **is_compiled**() const

void **finalize**()

void **perf_report**(std::ostream &os, std::size_t n, parameter_map params, std::size_t batch = 1, bool detailed = false) const

void **mark**(const parameter_map &params, marker m)

value **to_value**() const

void **from_value**(const value &v)

void **debug_print**() const

void **debug_print**(*instruction_ref* ins) const

void **print**(std::unordered_map<*instruction_ref*, std::string> &names, const std::function<void(*instruction_ref*, std::unordered_map<*instruction_ref*, std::string>)> &print_func) const

void **print**(const std::function<void(*instruction_ref* ins, std::unordered_map<*instruction_ref*, std::string>)> &print_func) const

void **print_graph**(std::ostream &os, bool brief = false) const

void **print_py**(std::ostream &os) const

void **print_cpp**(std::ostream &os) const

void **dry_run**(const parameter_map &params) const

void **annotate**(std::ostream &os, const std::function<void(*instruction_ref*)> &a) const

program &**sort**()

module ***create_module**(const std::string &name)

module ***create_module**(const std::string &name, module m) module ***get_module**(const std::string &name)

const module ***get_module**(const std::string &name) const

module ***get_main_module**()

const module ***get_main_module**() const

std::vector<const module*> **get_modules**() const

std::vector<module*> **get_modules**()

std::unordered_multimap<module_ref, module_ref> **get_module_tree**()

void **remove_module**(const std::string &name)

void **rename_module**(const std::string &old_name, const std::string &new_name)

void **remove_unused_modules**()

Friends

friend std::ostream &**operator<<**(std::ostream &os, const *program* &p)

friend bool **operator==**(const *program* &x, const *program* &y)

inline friend bool **operator!=**(const *program* &x, const *program* &y)

5.6.4 parse_onnx

program migraphx::internal::**parse_onnx**(const std::string &name, const *onnx_options*& = *onnx_options*{})

Create a program from an onnx file.

5.6.5 parse_tf

program migraphx::internal::**parse_tf**(const std::string &name, const *tf_options* &options = *tf_options*{})

Create a program from a tf pb file (default is nhwc format)

5.6.6 onnx_options

struct **onnx_options**

struct to pass in onnx options to parser

5.6.7 tf_options

struct **tf_options**

struct to pass in tf options to parser

5.7 Targets

5.7.1 target

struct **target**

An interface for a compilation target.

Public Functions

std::string **name**() const

A unique name used to identify the target.

std::vector<*pass*> **get_passes**(context &ctx, const compile_options &options) const

The transformation pass to be run during compilation.

Parameters

- **ctx** – This is the target-dependent context that is created by `get_context`
- **options** – Compiling options passed in by the user

Returns

The passes to be ran

context **get_context**() const

Construct a context for the target.

Returns

The context to be used during compilation and execution.

supported_segments **target_is_supported**(T&, const_module_ref mod, support_metric metric) const

Get the ranges of instructions that are supported on a target.

Parameters

- **module** – Module to check for supported instructions
- **metric** – Used to define how the quality of the support should be measured

Returns

the supported segments of the graph

argument **copy_to**(const *argument* &arg) const

copy an argument to the current target.

Parameters

arg – Input argument to be copied to the target

Returns

Argument in the target.

argument **copy_from**(const *argument* &arg) const

copy an argument from the current target.

Parameters

arg – Input argument to be copied from the target

Returns

Argument in the host.

argument **allocate**(const *shape* &s) const

Allocate an argument based on the input shape.

Parameters

s – Shape of the argument to be allocated in the target

Returns

Allocated argument in the target.

5.7.2 gpu::target

struct **target**

Public Functions

std::string **name**() const

std::vector<*pass*> **get_passes**(migraphx::context &gctx, const compile_options &options) const

migraphx::context **get_context**() const

argument **copy_to**(const *argument* &arg) const

argument **copy_from**(const *argument* &arg) const

argument **allocate**(const *shape* &s) const

5.7.3 cpu::target

struct **target**

Public Functions

std::string **name**() const

std::vector<*pass*> **get_passes**(migraphx::context &gctx, const compile_options&) const

inline migraphx::context **get_context**() const

inline *argument* **copy_to**(const *argument* &arg) const

inline *argument* **copy_from**(const *argument* &arg) const

argument **allocate**(const *shape* &s) const

5.8 Quantization

5.8.1 quantize_fp16

void migraphx::internal::**quantize_fp16**(*program* &prog, const std::vector<std::string> &ins_names = {"all"})

5.8.2 quantize_bf16

void migraphx::internal::**quantize_bf16**(*program* &prog, const std::vector<std::string> &ins_names = {"all"})

5.8.3 quantize_int8

void migraphx::internal::**quantize_int8**(*program* &prog, const *target* &t, const std::vector<parameter_map> &calibration, const std::unordered_set<std::string> &ins_names = {"dot", "convolution"})

5.9 Passes

5.9.1 pass

struct **pass**

An interface for applying a transformation to the instructions in a `program`

Public Functions

std::string **name**() const

A unique name used to identify the pass.

void **apply**(module_pass_manager &mpm) const

Run the pass on the module.

void **apply**(module &m) const

void **apply**(*program* &p) const

Run the pass on the program.

5.9.2 dead_code_elimination

struct **dead_code_elimination**

Remove instructions where the output is not used.

Public Functions

inline std::string **name**() const

void **apply**(module &m) const

void **apply**(*program* &p) const

5.9.3 eliminate_common_subexpression

struct **eliminate_common_subexpression**

Remove identical instructions.

Public Functions

inline std::string **name**() const

void **apply**(module &m) const

5.9.4 eliminate_concat

struct **eliminate_concat**

Remove concat operators by having each operator can write to different chunk of memory.

Public Functions

inline std::string **name**() const

void **apply**(module &m) const

Public Members

concat_optimization **concat_opt**

5.9.5 eliminate_contiguous

struct **eliminate_contiguous**

Remove contiguous instructions by checking if the operator can use non-standard shapes.

Public Functions

inline std::string **name**() const

void **apply**(module &m) const

Public Members

std::string **op_name**

5.9.6 eliminate_identity

struct **eliminate_identity**

Remove identity instructions. Currently when used as the last pass, it will preserve the semantics of previous program state, therefore dead code elimination should not be used afterwards.

Public Functions

inline std::string **name**() const

void **apply**(module &m) const

5.9.7 eliminate_pad

struct **eliminate_pad**

Remove pads if they can be written as an attribute to another op (im2col, convolution, pooling)

Public Functions

inline std::string **name**() const

void **apply**(module &m) const

5.9.8 propagate_constant

struct **propagate_constant**

Replace instructions which take all literals with a literal of the computation.

Public Functions

inline std::string **name**() const

void **apply**(module &m) const

Public Members

std::unordered_set<std::string> **skip_ops** = {}

5.9.9 rewrite_rnn

struct **rewrite_rnn**

Rewrite rnn to gemm and add.

Public Functions

inline std::string **name**() const

void **apply**(module &m) const

5.9.10 schedule

struct **schedule**

Schedule instructions for concurrent execution

Public Functions

inline std::string **name**() const

void **apply**(module &m) const

Public Members

schedule_model **model** = {}

bool **enable** = true

5.9.11 simplify_algebra

struct **simplify_algebra**

Simplify many algebraic instructions to more efficient versions.

Public Functions

inline std::string **name**() const

void **apply**(module &m) const

5.9.12 simplify_reshapes

struct **simplify_reshapes**

Eliminate redundant reshapes.

Public Functions

inline std::string **name**() const

void **apply**(module &m) const

Public Members

size_t **depth** = 4

bool **enable_op_shape_transform_op** = false

bool **enable_gather_rewrite** = false

5.10 Matchers

5.10.1 Introduction

The matchers provide a way to compose several predicates together. A matcher such as `m(m1, m2)` first checks a match for `m` followed by a match for `m1` and `m2` subsequently.

The most commonly used matcher is the `name` matcher. It matches the instruction with the operator equal to the name specified:

```
auto match_sum = name("sum");
```

The above matcher finds `sum` operators. To find `sum` operators with the output `standard_shape`, use:

auto match_sum = name("sum")(standard_shape());

5.10.2 Arguments

To match arguments in the instructions, match each argument using the `arg` matcher:

```
auto match_sum = name("sum")(arg(0)(name("@literal"), arg(1)(name("@literal"))));
```

The above matcher matches a `sum` operator with two arguments that are literals. Note that the `args` matcher eliminates the need to write `arg(0)` and `arg(1)` everytime:

```
auto match_sum = name("sum")(args(name("@literal"), name("@literal")));
```

5.10.3 Binding

To reference other instructions encountered while traversing through the instructions, use `.bind`:

```
auto match_sum = name("sum")(args(
                                name("@literal").bind("one"),
                                name("@literal").bind("two")
                            )).bind("sum");
```

This associates the instruction to a name that can be read from the `matcher_result` when it matches.

5.10.4 Finding matches

To use the matchers to find instructions, write a callback object that contains the matcher and an `apply` function that takes the `matcher_result` when the match is found:

```
struct match_find_sum
{
    auto matcher() const { return name("sum"); }

    void apply(program& p, matcher_result r) const
    {
        // Do something with the result
    }
};

find_matches(prog, match_find_sum{});
```

5.10.5 Creating matchers

The macros `MIGRAPH_BASIC_MATCHER` and `MIGRAPH_PRED_MATCHER` help in the creation of the matchers. Here is how you can create a matcher for shapes that are broadcasted:

```
MIGRAPH_PRED_MATCHER(broadcasted_shape, instruction_ref ins)
{
    return ins->get_shape().broadcasted();
}
```

For parameters to the predicate, use `make_basic_pred_matcher` to create the matcher. Here is how you can create a matcher to check the number of dimensions of the shape:

```
inline auto number_of_dims(std::size_t n)
{
    return make_basic_pred_matcher([=](instruction_ref ins) {
        return ins->get_shape().lens().size() == n;
    });
}
```

5.11 Tools

5.11.1 roctx.py

You can use the roctx command with rocprof binary to get marker timing information for each MIGraphX operator. To process timing information, use `roctx.py` helper script.

```
Usage: roctx.py [-h] [--json-path json_path] [--out out]
[--study-name study-name] [--repeat repeat] [--parse]
[--run run] [--debug]
```

The `roctx.py` helper script provides two main functionalities: `run` and `parse`.

`--run`

Runs `migraphx-driver roctx` command with the given `migraphx-driver` knobs followed by the parsing of the result which provides GPU kernel timing information. You can pass the MIGraphX knobs via a string to *–run* knob. See the _roctx-examples for usage.

`--parse`

Parses JSON file in the given `--json-path` and provides GPU kernel timing information.

`--out`

Output folder

`--study-name`

Optional. Allows user to name a study for easy interpretation. Defaults to timestamp.

`--repeat`

Number of iterations. Sets to **2** by default.

`--debug`

Provides additional debug information related to data. Use for debugging purposes only.

Examples:

Running inference with rocTX for a given ONNX file:

```
python roctx.py --run '--onnx --gpu fcn-resnet50-11.onnx' --out output_folder --repeat 5
```

Example output:

```
*** RESULTS ***
                                                    SUM_avg  MIN_avg  MAX_avg  COUNT
Marker start: gpu::convolution                         1625        8      103     31
Marker start: gpu::conv_bias_relu                      1212        8      450     18
Marker start: gpu::add_relu                             155        1       11     23
Marker start: gpu::conv_bias                            112       12       43      4
Marker start: load                                      110        0        2    160
Marker start: gpu::triadd_relu                           90        7       11     10
Marker start: hip::hip_copy_literal                      77        0        2    110
Marker start: broadcast                                  41        0        2     53
Marker start: gpu::concat                                39        6        7      6
Marker start: gpu::mul_add                               27        2        8      7
Marker start: gpu::sub                                   22        2        5      8
Marker start: slice                                      12        0        1     16
Marker start: gpu::pooling                                7        7        7      1
Marker start: step                                        3        3        3      1
Marker start: multibroadcast                              2        1        1      2
Marker start: @param                                      2        0        1      3
Marker start: hip::hip_allocate_memory                    1        1        1      1
Marker start: check_context::migraphx::version_...        0        0        0      0

AVG TOTAL TIME: 3544 us

OUTPUT CSV FILE:        output2021_11_04-03:03:02_AM.csv
KERNEL TIMING DETAILS:  rocTX_kernel_timing_details.txt
ALL DATA FROM ALL RUNS: rocTX_runs_dataframe.csv
```

Hotspot kernel timing information:

```
MOST TIME CONSUMING KERNELS IN EACH ITERATION (EXPECTED TO BE SAME KERNEL):
KERNEL NAME: miopenSp3AsmConv_v21_1_2_gfx9_fp32_stride1        448
KERNEL NAME: miopenSp3AsmConv_v21_1_2_gfx9_fp32_stride1        450
KERNEL NAME: miopenSp3AsmConv_v21_1_2_gfx9_fp32_stride1        449
KERNEL NAME: miopenSp3AsmConv_v21_1_2_gfx9_fp32_stride1        451
KERNEL NAME: miopenSp3AsmConv_v21_1_2_gfx9_fp32_stride1        456
```

The output provides `SUM`, `MIN`, `MAX` and `COUNT` information for each kernel executed for a given model. It also provides the average total time. The following three files are provided for reference:

- OUTPUT CSV FILE: Provides a summary of the run which includes utilized MIGraphX knobs and related kernel timing information.
- KERNEL TIMING DETAILS: Provides the hotspot kernel timing information.
- ALL DATA FROM ALL RUNS: Provides all output data related to all iterations executed during a run.

Parsing an existing JSON file:

```
python roctx.py --parse --json-path ../trace.json
```

CHAPTER

SIX

MIGRAPHX EXAMPLES

Example code for the following use cases is available in the MIGraphX GitHub repository:

- Diffusion inference
- AMD MIGraphX usage and utilities
- Natural language processing inference
- ONNX runtime
- Transformer inference
- Vision inference

CHAPTER

SEVEN

LICENSE

PYTHON MODULE INDEX

m

migraphx, 19

INDEX

Symbols

--debug
    command line option, 154
--out
    command line option, 154
--parse
    command line option, 154
--repeat
    command line option, 154
--run
    command line option, 154
--study-name
    command line option, 154

A

add_instruction() (*in module migraphx*), 23
add_literal() (*in module migraphx*), 23
add_parameter() (*in module migraphx*), 23
add_return() (*in module migraphx*), 23
argument (*class in migraphx*), 21
argument_from_pointer() (*in module migraphx*), 22
autocast_fp8() (*in module migraphx*), 25

B

broadcasted() (*in module migraphx*), 20
bytes() (*in module migraphx*), 20

C

clone() (*in module migraphx*), 23
command line option
    --debug, 154
    --out, 154
    --parse, 154
    --repeat, 154
    --run, 154
    --study-name, 154
compile() (*in module migraphx*), 24
create_argument() (*in module migraphx*), 22
create_module() (*in module migraphx*), 24

D

data_ptr() (*in module migraphx*), 21
dyn_dims() (*in module migraphx*), 20
dynamic() (*in module migraphx*), 20
dynamic_dimension (*class in migraphx*), 21

E

elements() (*in module migraphx*), 19

F

fill_argument() (*in module migraphx*), 21

G

generate_argument() (*in module migraphx*), 21
get_main_module() (*in module migraphx*), 24
get_output_shapes() (*in module migraphx*), 24
get_parameter_names() (*in module migraphx*), 23
get_parameter_shapes() (*in module migraphx*), 23
get_shape() (*in module migraphx*), 21
get_target() (*in module migraphx*), 22

I

is_fixed() (*in module migraphx*), 21

L

lens() (*in module migraphx*), 19
load() (*in module migraphx*), 26
load() (*migraphx.argument static method*), 22

M

migraphx
    module, 19
migraphx::argument (*C++ struct*), 14
migraphx::argument::argument (*C++ function*), 14
migraphx::argument::as_vector (*C++ function*), 14
migraphx::argument::data (*C++ function*), 14
migraphx::argument::generate (*C++ function*), 14
migraphx::argument::get_shape (*C++ function*), 14
migraphx::argument::load_argument (*C++ function*), 14
migraphx::argument::operator!= (*C++ function*), 14
migraphx::argument::operator== (*C++ function*), 14

migraphx::argument::save_argument (C++ *function*), 14
migraphx::file_options (C++ *struct*), 19
migraphx::file_options::file_options (C++ *function*), 19
migraphx::file_options::set_file_format (C++ *function*), 19
migraphx::internal::argument (C++ *struct*), 67
migraphx::internal::argument::argument (C++ *function*), 67, 68
migraphx::internal::argument::convert (C++ *function*), 68
migraphx::internal::argument::copy (C++ *function*), 68
migraphx::internal::argument::data (C++ *function*), 68
migraphx::internal::argument::element (C++ *function*), 68
migraphx::internal::argument::empty (C++ *function*), 68
migraphx::internal::argument::fill (C++ *function*), 68
migraphx::internal::argument::get_shape (C++ *function*), 68
migraphx::internal::argument::get_sub_objects (C++ *function*), 68
migraphx::internal::argument::reshape (C++ *function*), 68
migraphx::internal::argument::share (C++ *function*), 68
migraphx::internal::cpu::target (C++ *struct*), 148
migraphx::internal::cpu::target::allocate (C++ *function*), 148
migraphx::internal::cpu::target::copy_from (C++ *function*), 148
migraphx::internal::cpu::target::copy_to (C++ *function*), 148
migraphx::internal::cpu::target::get_context (C++ *function*), 148
migraphx::internal::cpu::target::get_passes (C++ *function*), 148
migraphx::internal::cpu::target::name (C++ *function*), 148
migraphx::internal::dead_code_elimination (C++ *struct*), 149
migraphx::internal::dead_code_elimination::apply (C++ *function*), 149
migraphx::internal::dead_code_elimination::name (C++ *function*), 149
migraphx::internal::eliminate_common_subexpression (C++ *struct*), 149
migraphx::internal::eliminate_common_subexpression::apply (C++ *function*), 149
migraphx::internal::eliminate_common_subexpression::name (C++ *function*), 149
migraphx::internal::eliminate_concat (C++ *struct*), 149
migraphx::internal::eliminate_concat::apply (C++ *function*), 150
migraphx::internal::eliminate_concat::concat_opt (C++ *member*), 150
migraphx::internal::eliminate_concat::name (C++ *function*), 150
migraphx::internal::eliminate_contiguous (C++ *struct*), 150
migraphx::internal::eliminate_contiguous::apply (C++ *function*), 150
migraphx::internal::eliminate_contiguous::name (C++ *function*), 150
migraphx::internal::eliminate_contiguous::op_name (C++ *member*), 150
migraphx::internal::eliminate_identity (C++ *struct*), 150
migraphx::internal::eliminate_identity::apply (C++ *function*), 150
migraphx::internal::eliminate_identity::name (C++ *function*), 150
migraphx::internal::eliminate_pad (C++ *struct*), 150
migraphx::internal::eliminate_pad::apply (C++ *function*), 150
migraphx::internal::eliminate_pad::name (C++ *function*), 150
migraphx::internal::gpu::target (C++ *struct*), 148
migraphx::internal::gpu::target::allocate (C++ *function*), 148
migraphx::internal::gpu::target::copy_from (C++ *function*), 148
migraphx::internal::gpu::target::copy_to (C++ *function*), 148
migraphx::internal::gpu::target::get_context (C++ *function*), 148
migraphx::internal::gpu::target::get_passes (C++ *function*), 148
migraphx::internal::gpu::target::name (C++ *function*), 148
migraphx::internal::has_finalize (C++ *function*), 73
migraphx::internal::instruction (C++ *struct*), 142
migraphx::internal::instruction::add_output (C++ *function*), 143
migraphx::internal::instruction::backreference (C++ *function*), 143
migraphx::internal::instruction::can_eval (C++ *function*), 143

migraphx::internal::instruction::clear_arguments (C++ *function*), 142
migraphx::internal::instruction::debug_print (C++ *function*), 143
migraphx::internal::instruction::eval (C++ *function*), 143
migraphx::internal::instruction::finalize (C++ *function*), 143
migraphx::internal::instruction::get_literal (C++ *function*), 143
migraphx::internal::instruction::get_operator (C++ *function*), 143
migraphx::internal::instruction::get_output_alias (C++ *function*), 143
migraphx::internal::instruction::get_shape (C++ *function*), 143
migraphx::internal::instruction::get_target_id (C++ *function*), 143
migraphx::internal::instruction::inputs (C++ *function*), 143
migraphx::internal::instruction::instruction (C++ *function*), 142
migraphx::internal::instruction::is_normalized (C++ *function*), 143
migraphx::internal::instruction::is_undefined (C++ *function*), 143
migraphx::internal::instruction::module_inputs (C++ *function*), 143
migraphx::internal::instruction::name (C++ *function*), 143
migraphx::internal::instruction::need_normalization (C++ *function*), 143
migraphx::internal::instruction::normalized_operator (C++ *function*), 143
migraphx::internal::instruction::operator!= (C++ *function*), 144
migraphx::internal::instruction::operator== (C++ *function*), 144
migraphx::internal::instruction::outputs (C++ *function*), 143
migraphx::internal::instruction::print (C++ *function*), 144
migraphx::internal::instruction::recompute_shape (C++ *function*), 142
migraphx::internal::instruction::remove_output (C++ *function*), 143
migraphx::internal::instruction::replace (C++ *function*), 142, 143
migraphx::internal::instruction::replace_argument (C++ *function*), 143
migraphx::internal::instruction::replace_mod_argument (C++ *function*), 143
migraphx::internal::instruction::replace_refs (C++ *function*), 143
migraphx::internal::instruction::set_normalized (C++ *function*), 143
migraphx::internal::instruction::set_target_id (C++ *function*), 143
migraphx::internal::instruction::valid (C++ *function*), 142
migraphx::internal::instruction_ref (C++ *type*), 144
migraphx::internal::is_context_free (C++ *function*), 73
migraphx::internal::onnx_options (C++ *struct*), 146
migraphx::internal::operation (C++ *struct*), 72
migraphx::internal::operation::compute (C++ *function*), 73
migraphx::internal::operation::compute_shape (C++ *function*), 72
migraphx::internal::operation::finalize (C++ *function*), 72
migraphx::internal::operation::name (C++ *function*), 72
migraphx::internal::operation::operator<< (C++ *function*), 73
migraphx::internal::operation::output_alias (C++ *function*), 73
migraphx::internal::parse_onnx (C++ *function*), 146
migraphx::internal::parse_tf (C++ *function*), 146
migraphx::internal::program (C++ *struct*), 144
migraphx::internal::program::~program (C++ *function*), 144
migraphx::internal::program::annotate (C++ *function*), 145
migraphx::internal::program::compile (C++ *function*), 145
migraphx::internal::program::create_module (C++ *function*), 145
migraphx::internal::program::debug_print (C++ *function*), 145
migraphx::internal::program::dry_run (C++ *function*), 145
migraphx::internal::program::eval (C++ *function*), 144
migraphx::internal::program::eval_with_context (C++ *function*), 144
migraphx::internal::program::finalize (C++ *function*), 145
migraphx::internal::program::finish (C++ *function*), 145
migraphx::internal::program::from_value (C++ *function*), 145
migraphx::internal::program::get_context (C++ *function*), 145
migraphx::internal::program::get_main_module

(C++ function), 146
migraphx::internal::program::get_module (C++ function), 145, 146
migraphx::internal::program::get_module_tree (C++ function), 146
migraphx::internal::program::get_modules (C++ function), 146
migraphx::internal::program::get_output_shapes (C++ function), 145
migraphx::internal::program::get_parameter (C++ function), 144
migraphx::internal::program::get_parameter_names (C++ function), 144
migraphx::internal::program::get_parameter_shape (C++ function), 144
migraphx::internal::program::get_parameter_shapes (C++ function), 144
migraphx::internal::program::get_target_assignments (C++ function), 145
migraphx::internal::program::is_compiled (C++ function), 145
migraphx::internal::program::mark (C++ function), 145
migraphx::internal::program::operator!= (C++ function), 146
migraphx::internal::program::operator= (C++ function), 144
migraphx::internal::program::operator== (C++ function), 146
migraphx::internal::program::operator<< (C++ function), 146
migraphx::internal::program::perf_report (C++ function), 145
migraphx::internal::program::print (C++ function), 145
migraphx::internal::program::print_cpp (C++ function), 145
migraphx::internal::program::print_graph (C++ function), 145
migraphx::internal::program::print_py (C++ function), 145
migraphx::internal::program::program (C++ function), 144
migraphx::internal::program::remove_module (C++ function), 146
migraphx::internal::program::remove_unused_modules (C++ function), 146
migraphx::internal::program::rename_module (C++ function), 146
migraphx::internal::program::size (C++ function), 145
migraphx::internal::program::sort (C++ function), 145
migraphx::internal::program::to_value (C++ function), 145
migraphx::internal::program::total_instructions (C++ function), 144
migraphx::internal::program::validate (C++ function), 145
migraphx::internal::quantize_bf16 (C++ function), 148
migraphx::internal::quantize_fp16 (C++ function), 148
migraphx::internal::quantize_int8 (C++ function), 148
migraphx::internal::shape (C++ struct), 59
migraphx::internal::shape::any_of_dynamic (C++ function), 62
migraphx::internal::shape::as (C++ struct), 63
migraphx::internal::shape::as::epsilon (C++ function), 64
migraphx::internal::shape::as::from (C++ function), 64
migraphx::internal::shape::as::is_integral (C++ function), 64
migraphx::internal::shape::as::is_signed (C++ function), 64
migraphx::internal::shape::as::is_unsigned (C++ function), 64
migraphx::internal::shape::as::max (C++ function), 64
migraphx::internal::shape::as::min (C++ function), 64
migraphx::internal::shape::as::nan (C++ function), 64
migraphx::internal::shape::as::operator() (C++ function), 64
migraphx::internal::shape::as::size (C++ function), 64
migraphx::internal::shape::as::type (C++ type), 64
migraphx::internal::shape::as::type_enum (C++ function), 64
migraphx::internal::shape::as_standard (C++ function), 62
migraphx::internal::shape::broadcasted (C++ function), 62
migraphx::internal::shape::bytes (C++ function), 61
migraphx::internal::shape::computable (C++ function), 62
migraphx::internal::shape::cpp_type (C++ function), 63
migraphx::internal::shape::dyn_dims (C++ function), 61
migraphx::internal::shape::dynamic (C++ function), 62
migraphx::internal::shape::dynamic_dimension

(*C++ struct*), 64
migraphx::internal::shape::dynamic_dimension::has_optimal (*C++ function*), 64
migraphx::internal::shape::dynamic_dimension::intersection (*C++ function*), 64
migraphx::internal::shape::dynamic_dimension::is_fixed (*C++ function*), 64
migraphx::internal::shape::dynamic_dimension::max (*C++ member*), 65
migraphx::internal::shape::dynamic_dimension::min (*C++ member*), 65
migraphx::internal::shape::dynamic_dimension::operator!= (*C++ function*), 65
migraphx::internal::shape::dynamic_dimension::operator* (*C++ function*), 65
migraphx::internal::shape::dynamic_dimension::operator*= (*C++ function*), 64
migraphx::internal::shape::dynamic_dimension::operator+ (*C++ function*), 65
migraphx::internal::shape::dynamic_dimension::operator+= (*C++ function*), 64
migraphx::internal::shape::dynamic_dimension::operator== (*C++ function*), 65
migraphx::internal::shape::dynamic_dimension::operator- (*C++ function*), 65
migraphx::internal::shape::dynamic_dimension::operator-= (*C++ function*), 64
migraphx::internal::shape::dynamic_dimension::operator<< (*C++ function*), 65
migraphx::internal::shape::dynamic_dimension::optimals (*C++ member*), 65
migraphx::internal::shape::dynamic_dimension::reflect (*C++ function*), 65
migraphx::internal::shape::element_space (*C++ function*), 63
migraphx::internal::shape::elements (*C++ function*), 61
migraphx::internal::shape::from_permutation (*C++ function*), 63
migraphx::internal::shape::get_type (*C++ struct*), 65
migraphx::internal::shape::get_type<bf16, T> (*C++ struct*), 65
migraphx::internal::shape::get_type<bool, T> (*C++ struct*), 65
migraphx::internal::shape::get_type<const T> (*C++ struct*), 65
migraphx::internal::shape::get_type<double, T> (*C++ struct*), 65
migraphx::internal::shape::get_type<float, T> (*C++ struct*), 66
migraphx::internal::shape::get_type<half, T> (*C++ struct*), 66
migraphx::internal::shape::get_type<int16_t, T> (*C++ struct*), 66
migraphx::internal::shape::get_type<int32_t, T> (*C++ struct*), 66
migraphx::internal::shape::get_type<int64_t, T> (*C++ struct*), 66
migraphx::internal::shape::get_type<int8_t, T> (*C++ struct*), 66
migraphx::internal::shape::get_type<migraphx::fp8::fp8e4m3 T> (*C++ struct*), 66
migraphx::internal::shape::get_type<migraphx::fp8::fp8e4m3 T> (*C++ struct*), 66
migraphx::internal::shape::get_type<migraphx::fp8::fp8e5m2 T> (*C++ struct*), 66
migraphx::internal::shape::get_type<migraphx::fp8::fp8e5m2 T> (*C++ struct*), 66
migraphx::internal::shape::get_type<uint16_t, T> (*C++ struct*), 66
migraphx::internal::shape::get_type<uint32_t, T> (*C++ struct*), 66
migraphx::internal::shape::get_type<uint64_t, T> (*C++ struct*), 66
migraphx::internal::shape::get_type<uint8_t, T> (*C++ struct*), 66
migraphx::internal::shape::index (*C++ function*), 61
migraphx::internal::shape::is_compatible (*C++ function*), 63
migraphx::internal::shape::is_computable (*C++ function*), 63
migraphx::internal::shape::is_integral (*C++ function*), 63
migraphx::internal::shape::is_unsigned (*C++ function*), 63
migraphx::internal::shape::lens (*C++ function*), 61
migraphx::internal::shape::max_lens (*C++ function*), 61
migraphx::internal::shape::min_lens (*C++ function*), 61
migraphx::internal::shape::multi (*C++ function*), 62
migraphx::internal::shape::multi_copy (*C++ function*), 62
migraphx::internal::shape::multi_within_bounds (*C++ function*), 62
migraphx::internal::shape::name (*C++ function*), 63
migraphx::internal::shape::ndim (*C++ function*), 61
migraphx::internal::shape::normalize_standard (*C++ function*), 62
migraphx::internal::shape::operator!= (*C++ function*), 63
migraphx::internal::shape::operator== (*C++

function), 63
migraphx::internal::shape::operator<< (C++ function), 63
migraphx::internal::shape::opt_lens (C++ function), 61
migraphx::internal::shape::packed (C++ function), 62
migraphx::internal::shape::parse_type (C++ function), 63
migraphx::internal::shape::scalar (C++ function), 62
migraphx::internal::shape::shape (C++ function), 60, 61
migraphx::internal::shape::single (C++ function), 62
migraphx::internal::shape::standard (C++ function), 62
migraphx::internal::shape::strides (C++ function), 61
migraphx::internal::shape::sub_shapes (C++ function), 63
migraphx::internal::shape::to_dynamic (C++ function), 62
migraphx::internal::shape::to_sizes_string (C++ function), 63
migraphx::internal::shape::to_static (C++ function), 62
migraphx::internal::shape::transposed (C++ function), 62
migraphx::internal::shape::tuple_size (C++ function), 63
migraphx::internal::shape::type (C++ function), 61
migraphx::internal::shape::type_size (C++ function), 61
migraphx::internal::shape::type_string (C++ function), 62
migraphx::internal::shape::type_t (C++ enum), 59
migraphx::internal::shape::type_t::bf16_type (C++ enumerator), 60
migraphx::internal::shape::type_t::bool_type (C++ enumerator), 59
migraphx::internal::shape::type_t::double_type (C++ enumerator), 60
migraphx::internal::shape::type_t::float_type (C++ enumerator), 60
migraphx::internal::shape::type_t::fp4x2_type (C++ enumerator), 60
migraphx::internal::shape::type_t::fp8e4m3fn_type (C++ enumerator), 60
migraphx::internal::shape::type_t::fp8e4m3fnuz_type (C++ enumerator), 60
migraphx::internal::shape::type_t::fp8e5m2_type (C++ enumerator), 60
migraphx::internal::shape::type_t::fp8e5m2fnuz_type (C++ enumerator), 60
migraphx::internal::shape::type_t::half_type (C++ enumerator), 60
migraphx::internal::shape::type_t::int16_type (C++ enumerator), 60
migraphx::internal::shape::type_t::int32_type (C++ enumerator), 60
migraphx::internal::shape::type_t::int64_type (C++ enumerator), 60
migraphx::internal::shape::type_t::int8_type (C++ enumerator), 60
migraphx::internal::shape::type_t::tuple_type (C++ enumerator), 60
migraphx::internal::shape::type_t::uint16_type (C++ enumerator), 60
migraphx::internal::shape::type_t::uint32_type (C++ enumerator), 60
migraphx::internal::shape::type_t::uint64_type (C++ enumerator), 60
migraphx::internal::shape::type_t::uint8_type (C++ enumerator), 60
migraphx::internal::shape::types (C++ function), 63
migraphx::internal::shape::visit (C++ function), 63
migraphx::internal::shape::visit_type (C++ function), 62
migraphx::internal::shape::visit_types (C++ function), 63
migraphx::internal::shape::with_lens (C++ function), 62
migraphx::internal::shape::with_type (C++ function), 62
migraphx::internal::target (C++ struct), 146
migraphx::internal::target::allocate (C++ function), 147
migraphx::internal::target::copy_from (C++ function), 147
migraphx::internal::target::copy_to (C++ function), 147
migraphx::internal::target::get_context (C++ function), 147
migraphx::internal::target::get_passes (C++ function), 147
migraphx::internal::target::name (C++ function), 147
migraphx::internal::target::target_is_supported (C++ function), 147
migraphx::internal::tensor_view (C++ struct), 71
migraphx::internal::tensor_view::back (C++ function), 72

migraphx::internal::tensor_view::begin (*C++ function*), 72
migraphx::internal::tensor_view::begin_at (*C++ function*), 72
migraphx::internal::tensor_view::const_iterator (*C++ type*), 71
migraphx::internal::tensor_view::data (*C++ function*), 71
migraphx::internal::tensor_view::empty (*C++ function*), 71
migraphx::internal::tensor_view::end (*C++ function*), 72
migraphx::internal::tensor_view::front (*C++ function*), 72
migraphx::internal::tensor_view::get_shape (*C++ function*), 71
migraphx::internal::tensor_view::iterator (*C++ type*), 71
migraphx::internal::tensor_view::operator() (*C++ function*), 71
migraphx::internal::tensor_view::operator<< (*C++ function*), 72
migraphx::internal::tensor_view::operator[] (*C++ function*), 71, 72
migraphx::internal::tensor_view::size (*C++ function*), 71
migraphx::internal::tensor_view::slice_at (*C++ function*), 72
migraphx::internal::tensor_view::tensor_view (*C++ function*), 71
migraphx::internal::tensor_view::to_vector (*C++ function*), 72
migraphx::internal::tensor_view::value_type (*C++ type*), 71
migraphx::internal::tf_options (*C++ struct*), 146
migraphx::internal::visit_all (*C++ function*), 70
migraphx::literal (*C++ struct*), 67
migraphx::literal::data (*C++ function*), 67
migraphx::literal::empty (*C++ function*), 67
migraphx::literal::get_argument (*C++ function*), 67
migraphx::literal::get_shape (*C++ function*), 67
migraphx::literal::get_sub_objects (*C++ function*), 67
migraphx::literal::literal (*C++ function*), 67
migraphx::load (*C++ function*), 19
migraphx::onnx_options (*C++ struct*), 18
migraphx::onnx_options::onnx_options (*C++ function*), 18
migraphx::onnx_options::set_default_dim_value (*C++ function*), 18
migraphx::onnx_options::set_default_dyn_dim_value (*C++ function*), 18
migraphx::onnx_options::set_default_loop_iterations (*C++ function*), 18
migraphx::onnx_options::set_dyn_input_parameter_shape (*C++ function*), 18
migraphx::onnx_options::set_external_data_path (*C++ function*), 18
migraphx::onnx_options::set_input_parameter_shape (*C++ function*), 18
migraphx::onnx_options::set_limit_loop_iterations (*C++ function*), 18
migraphx::op (*C++ type*), 73
migraphx::op::abs (*C++ struct*), 75
migraphx::op::abs::apply (*C++ function*), 75
migraphx::op::acos (*C++ struct*), 75
migraphx::op::acos::apply (*C++ function*), 75
migraphx::op::acosh (*C++ struct*), 75
migraphx::op::acosh::apply (*C++ function*), 75
migraphx::op::add (*C++ struct*), 75
migraphx::op::add::apply (*C++ function*), 75
migraphx::op::add::attributes (*C++ function*), 75
migraphx::op::add::point_function (*C++ function*), 75
migraphx::op::allocate (*C++ struct*), 75
migraphx::op::allocate::buf_type (*C++ member*), 76
migraphx::op::allocate::compute (*C++ function*), 76
migraphx::op::allocate::compute_shape (*C++ function*), 76
migraphx::op::allocate::name (*C++ function*), 76
migraphx::op::allocate::reflect (*C++ function*), 76
migraphx::op::allocate::s (*C++ member*), 76
migraphx::op::argmax (*C++ struct*), 76
migraphx::op::argmax::attributes (*C++ function*), 76
migraphx::op::argmax::axis (*C++ member*), 76
migraphx::op::argmax::calc_argmax (*C++ function*), 76
migraphx::op::argmax::compute (*C++ function*), 76
migraphx::op::argmax::name (*C++ function*), 76
migraphx::op::argmax::normalize_compute_shape (*C++ function*), 76
migraphx::op::argmax::reflect (*C++ function*), 77
migraphx::op::argmax::select_last_index (*C++ member*), 76
migraphx::op::argmin (*C++ struct*), 77
migraphx::op::argmin::attributes (*C++ function*), 77
migraphx::op::argmin::axis (*C++ member*), 77
migraphx::op::argmin::calc_argmin (*C++ function*), 77
migraphx::op::argmin::compute (*C++ function*), 77
migraphx::op::argmin::name (*C++ function*), 77

migraphx::op::argmin::normalize_compute_shape (C++ function), 77
migraphx::op::argmin::reflect (C++ function), 77
migraphx::op::argmin::select_last_index (C++ member), 77
migraphx::op::as_shape (C++ struct), 77
migraphx::op::as_shape::compute (C++ function), 77
migraphx::op::as_shape::compute_shape (C++ function), 77
migraphx::op::as_shape::name (C++ function), 77
migraphx::op::as_shape::output_alias (C++ function), 77
migraphx::op::as_shape::reflect (C++ function), 78
migraphx::op::as_shape::s (C++ member), 77
migraphx::op::asin (C++ struct), 78
migraphx::op::asin::apply (C++ function), 78
migraphx::op::asinh (C++ struct), 78
migraphx::op::asinh::apply (C++ function), 78
migraphx::op::atan (C++ struct), 78
migraphx::op::atan::apply (C++ function), 78
migraphx::op::atanh (C++ struct), 78
migraphx::op::atanh::apply (C++ function), 78
migraphx::op::binary (C++ struct), 78
migraphx::op::binary::attributes (C++ function), 78
migraphx::op::binary::base_attributes (C++ function), 78
migraphx::op::binary::compute (C++ function), 78
migraphx::op::binary::compute_shape (C++ function), 78
migraphx::op::binary::point_function (C++ function), 78
migraphx::op::binary::point_op (C++ function), 78
migraphx::op::bit_cast (C++ struct), 78
migraphx::op::bit_cast::compute (C++ function), 79
migraphx::op::bit_cast::compute_shape (C++ function), 79
migraphx::op::bit_cast::point_op (C++ function), 79
migraphx::op::bit_cast::reflect (C++ function), 79
migraphx::op::bit_cast::target_type (C++ member), 79
migraphx::op::bitwise_and (C++ struct), 79
migraphx::op::bitwise_and::apply (C++ function), 79
migraphx::op::bitwise_and::attributes (C++ function), 79
migraphx::op::bitwise_and::compute_shape (C++ function), 79
migraphx::op::bitwise_and::point_function (C++ function), 79
migraphx::op::broadcast (C++ struct), 79
migraphx::op::broadcast::attributes (C++ function), 79
migraphx::op::broadcast::axis (C++ member), 80
migraphx::op::broadcast::broadcast_lens (C++ member), 80
migraphx::op::broadcast::compute (C++ function), 79
migraphx::op::broadcast::compute_shape (C++ function), 79
migraphx::op::broadcast::name (C++ function), 79
migraphx::op::broadcast::output_alias (C++ function), 79
migraphx::op::broadcast::reflect (C++ function), 80
migraphx::op::broadcast_for_dot (C++ struct), 80
migraphx::op::broadcast_for_dot::attributes (C++ function), 80
migraphx::op::broadcast_for_dot::compute (C++ function), 80
migraphx::op::broadcast_for_dot::compute_shape (C++ function), 80
migraphx::op::broadcast_for_dot::name (C++ function), 80
migraphx::op::broadcast_for_dot::output_alias (C++ function), 80
migraphx::op::broadcast_with_dims (C++ struct), 80
migraphx::op::broadcast_with_dims::attributes (C++ function), 80
migraphx::op::broadcast_with_dims::compute (C++ function), 80
migraphx::op::broadcast_with_dims::compute_shape (C++ function), 80
migraphx::op::broadcast_with_dims::name (C++ function), 80
migraphx::op::builder (C++ type), 134
migraphx::op::builder::add (C++ function), 135
migraphx::op::builder::batchnorm (C++ struct), 135
migraphx::op::builder::batchnorm::epsilon (C++ member), 135
migraphx::op::builder::batchnorm::insert (C++ function), 135
migraphx::op::builder::batchnorm::reflect (C++ function), 135
migraphx::op::builder::builder_func (C++ type), 134
migraphx::op::builder::builder_map (C++ function), 135
migraphx::op::builder::celu (C++ struct), 135

migraphx::op::builder::celu::alpha (*C++ member*), 136
migraphx::op::builder::celu::insert (*C++ function*), 136
migraphx::op::builder::celu::reflect (*C++ function*), 136
migraphx::op::builder::clip (*C++ struct*), 136
migraphx::op::builder::clip::insert (*C++ function*), 136
migraphx::op::builder::clip::reflect (*C++ function*), 136
migraphx::op::builder::convolution (*C++ struct*), 136
migraphx::op::builder::convolution::add_bias (*C++ function*), 136
migraphx::op::builder::convolution::insert (*C++ function*), 136
migraphx::op::builder::convolution::name (*C++ function*), 136
migraphx::op::builder::convolution_base (*C++ struct*), 136
migraphx::op::builder::convolution_base::auto_pad (*C++ member*), 137
migraphx::op::builder::convolution_base::dilations (*C++ member*), 137
migraphx::op::builder::convolution_base::group (*C++ member*), 137
migraphx::op::builder::convolution_base::is_dynamic (*C++ function*), 136
migraphx::op::builder::convolution_base::make_conv_op (*C++ function*), 136
migraphx::op::builder::convolution_base::padding_mode (*C++ member*), 137
migraphx::op::builder::convolution_base::paddings (*C++ member*), 137
migraphx::op::builder::convolution_base::reflect (*C++ function*), 137
migraphx::op::builder::convolution_base::strides (*C++ member*), 137
migraphx::op::builder::convolution_base::validate_or_init_attributes (*C++ function*), 136
migraphx::op::builder::einsum (*C++ struct*), 137
migraphx::op::builder::einsum::analyze_equation (*C++ function*), 137
migraphx::op::builder::einsum::apply_broadcast_op (*C++ function*), 138
migraphx::op::builder::einsum::apply_reduce_sum_op (*C++ function*), 138
migraphx::op::builder::einsum::apply_transpose_op (*C++ function*), 138
migraphx::op::builder::einsum::arange (*C++ function*), 138
migraphx::op::builder::einsum::batch_dot (*C++ function*), 138
migraphx::op::builder::einsum::calc_dim (*C++ function*), 138
migraphx::op::builder::einsum::concat_vectors (*C++ function*), 138
migraphx::op::builder::einsum::equation (*C++ member*), 139
migraphx::op::builder::einsum::equation_info (*C++ struct*), 139
migraphx::op::builder::einsum::equation_info::duplicates (*C++ member*), 139
migraphx::op::builder::einsum::equation_info::ellipsis_ndi (*C++ member*), 139
migraphx::op::builder::einsum::equation_info::explicit_for (*C++ member*), 139
migraphx::op::builder::einsum::equation_info::input_terms (*C++ member*), 139
migraphx::op::builder::einsum::equation_info::label_count (*C++ member*), 139
migraphx::op::builder::einsum::equation_info::output_term (*C++ member*), 139
migraphx::op::builder::einsum::extract_column (*C++ function*), 138
migraphx::op::builder::einsum::finalize_output (*C++ function*), 138
migraphx::op::builder::einsum::find_duplicates (*C++ function*), 137
migraphx::op::builder::einsum::gather_diagonal (*C++ function*), 138
migraphx::op::builder::einsum::generate_output_term (*C++ function*), 137
migraphx::op::builder::einsum::insert (*C++ function*), 137
migraphx::op::builder::einsum::int_mat (*C++ type*), 137
migraphx::op::builder::einsum::make_mapping_matrix (*C++ function*), 137
migraphx::op::builder::einsum::make_matrix (*C++ function*), 138
migraphx::op::builder::einsum::parse_equation (*C++ function*), 137
migraphx::op::builder::einsum::preprocess_input (*C++ function*), 137
migraphx::op::builder::einsum::process_pair (*C++ function*), 138
migraphx::op::builder::einsum::reflect (*C++ function*), 139
migraphx::op::builder::einsum::set_difference (*C++ function*), 138
migraphx::op::builder::einsum::set_union (*C++ function*), 138
migraphx::op::builder::einsum::squeeze_transpose (*C++ function*), 138
migraphx::op::builder::einsum::transpose_unsqueeze (*C++ function*), 138

`migraphx::op::builder::einsum::validate_input_terms` (*C++ function*), 137
`migraphx::op::builder::einsum::validate_output_term` (*C++ function*), 137
`migraphx::op::builder::gelu_erf` (*C++ struct*), 139
`migraphx::op::builder::gelu_erf::insert` (*C++ function*), 139
`migraphx::op::builder::gelu_erf::reflect` (*C++ function*), 139
`migraphx::op::builder::gelu_quick` (*C++ struct*), 139
`migraphx::op::builder::gelu_quick::alpha` (*C++ member*), 139
`migraphx::op::builder::gelu_quick::insert` (*C++ function*), 139
`migraphx::op::builder::gelu_quick::reflect` (*C++ function*), 140
`migraphx::op::builder::gelu_split` (*C++ struct*), 140
`migraphx::op::builder::gelu_split::insert` (*C++ function*), 140
`migraphx::op::builder::gelu_split::reflect` (*C++ function*), 140
`migraphx::op::builder::gelu_tanh` (*C++ struct*), 140
`migraphx::op::builder::gelu_tanh::fast` (*C++ member*), 140
`migraphx::op::builder::gelu_tanh::insert` (*C++ function*), 140
`migraphx::op::builder::gelu_tanh::reflect` (*C++ function*), 140
`migraphx::op::builder::gemm` (*C++ struct*), 140
`migraphx::op::builder::gemm::alpha` (*C++ member*), 140
`migraphx::op::builder::gemm::beta` (*C++ member*), 140
`migraphx::op::builder::gemm::insert` (*C++ function*), 140
`migraphx::op::builder::gemm::reflect` (*C++ function*), 141
`migraphx::op::builder::gemm::trans_a` (*C++ member*), 140
`migraphx::op::builder::gemm::trans_b` (*C++ member*), 140
`migraphx::op::builder::insert` (*C++ function*), 135
`migraphx::op::builder::insert_common_op` (*C++ function*), 135
`migraphx::op::builder::invoke_builder` (*C++ function*), 135
`migraphx::op::builder::mean_variance_normalization` (*C++ struct*), 141
`migraphx::op::builder::mean_variance_normalization::axes` (*C++ member*), 141
`migraphx::op::builder::mean_variance_normalization::insert` (*C++ function*), 141
`migraphx::op::builder::mean_variance_normalization::reflec` (*C++ function*), 141
`migraphx::op::builder::op_builder` (*C++ struct*), 141
`migraphx::op::builder::op_builder::name` (*C++ function*), 141
`migraphx::op::builder::quant_convolution` (*C++ struct*), 141
`migraphx::op::builder::quant_convolution::add_int8_shift` (*C++ function*), 141
`migraphx::op::builder::quant_convolution::gen_symmetric_li` (*C++ function*), 141
`migraphx::op::builder::quant_convolution::get_symmetric_va` (*C++ function*), 141
`migraphx::op::builder::quant_convolution::get_zero_point` (*C++ function*), 141
`migraphx::op::builder::quant_convolution::handle_quant_bia` (*C++ function*), 142
`migraphx::op::builder::quant_convolution::handle_quant_inp` (*C++ function*), 142
`migraphx::op::builder::quant_convolution::insert` (*C++ function*), 141
`migraphx::op::builder::quant_convolution::is_symmetric_zer` (*C++ function*), 142
`migraphx::op::builder::quant_convolution::name` (*C++ function*), 142
`migraphx::op::builder::quant_convolution::qparam_broadcast` (*C++ function*), 142
`migraphx::op::builder::quant_convolution::shift_input_and_` (*C++ function*), 141
`migraphx::op::builder::register_builder` (*C++ function*), 135
`migraphx::op::builder::register_builder_action` (*C++ struct*), 142
`migraphx::op::builder::register_builder_action::apply` (*C++ function*), 142
`migraphx::op::cache_parameters` (*C++ struct*), 80
`migraphx::op::cache_parameters::batch_size` (*C++ member*), 81
`migraphx::op::cache_parameters::head_size` (*C++ member*), 81
`migraphx::op::cache_parameters::seqlen_present_kv_cache` (*C++ member*), 81
`migraphx::op::cache_parameters::sequence_length` (*C++ member*), 81
`migraphx::op::capture` (*C++ struct*), 81
`migraphx::op::capture::compute` (*C++ function*), 81
`migraphx::op::capture::compute_shape` (*C++ function*), 81
`migraphx::op::capture::f` (*C++ member*), 81

migraphx::op::capture::ins_index (*C++ member*), 81
migraphx::op::capture::name (*C++ function*), 81
migraphx::op::capture::output_alias (*C++ function*), 81
migraphx::op::capture::reflect (*C++ function*), 81
migraphx::op::ceil (*C++ struct*), 81
migraphx::op::ceil::apply (*C++ function*), 81
migraphx::op::clip (*C++ struct*), 81
migraphx::op::clip::attributes (*C++ function*), 81
migraphx::op::clip::compute (*C++ function*), 82
migraphx::op::clip::compute_shape (*C++ function*), 81
migraphx::op::clip::name (*C++ function*), 81
migraphx::op::concat (*C++ struct*), 82
migraphx::op::concat::attributes (*C++ function*), 82
migraphx::op::concat::axis (*C++ member*), 82
migraphx::op::concat::compute (*C++ function*), 82
migraphx::op::concat::compute_offsets (*C++ function*), 82
migraphx::op::concat::name (*C++ function*), 82
migraphx::op::concat::normalize_compute_shape (*C++ function*), 82
migraphx::op::concat::reflect (*C++ function*), 82
migraphx::op::concat_past_present (*C++ struct*), 82
migraphx::op::concat_past_present::compute (*C++ function*), 82
migraphx::op::concat_past_present::compute_shape (*C++ function*), 82
migraphx::op::concat_past_present::concat_state_chunk (*C++ function*), 82
migraphx::op::concat_past_present::copy_data (*C++ function*), 82
migraphx::op::concat_past_present::kv_num_heads (*C++ member*), 83
migraphx::op::concat_past_present::name (*C++ function*), 82
migraphx::op::concat_past_present::reflect (*C++ function*), 83
migraphx::op::concat_past_present::update_cache (*C++ function*), 82
migraphx::op::contiguous (*C++ struct*), 83
migraphx::op::contiguous::apply (*C++ function*), 83
migraphx::op::contiguous::compute (*C++ function*), 83
migraphx::op::contiguous::compute_shape (*C++ function*), 83
migraphx::op::contiguous::name (*C++ function*), 83
migraphx::op::convert (*C++ struct*), 83
migraphx::op::convert::apply (*C++ function*), 83
migraphx::op::convert::compute_shape (*C++ function*), 83
migraphx::op::convert::convert (*C++ function*), 83
migraphx::op::convert::point_op (*C++ function*), 83
migraphx::op::convert::reflect (*C++ function*), 83
migraphx::op::convert::target_type (*C++ member*), 83
migraphx::op::convolution (*C++ struct*), 83
migraphx::op::convolution::attributes (*C++ function*), 84
migraphx::op::convolution::calc_conv_lens (*C++ function*), 84
migraphx::op::convolution::check_attribute_size (*C++ function*), 84
migraphx::op::convolution::compute (*C++ function*), 84
migraphx::op::convolution::dilation (*C++ member*), 84
migraphx::op::convolution::dynamic_compute_shape (*C++ function*), 84
migraphx::op::convolution::group (*C++ member*), 84
migraphx::op::convolution::kdims (*C++ function*), 84
migraphx::op::convolution::name (*C++ function*), 84
migraphx::op::convolution::normalize_compute_shape (*C++ function*), 84
migraphx::op::convolution::padding (*C++ member*), 84
migraphx::op::convolution::padding_mode (*C++ member*), 84
migraphx::op::convolution::reflect (*C++ function*), 84
migraphx::op::convolution::static_compute_shape (*C++ function*), 84
migraphx::op::convolution::stride (*C++ member*), 84
migraphx::op::convolution_backwards (*C++ struct*), 84
migraphx::op::convolution_backwards::calc_spatial_lens (*C++ function*), 84
migraphx::op::convolution_backwards::check_attribute_size (*C++ function*), 84
migraphx::op::convolution_backwards::compute (*C++ function*), 85
migraphx::op::convolution_backwards::compute_shape (*C++ function*), 84
migraphx::op::convolution_backwards::dilation

(C++ member), 85
migraphx::op::convolution_backwards::dynamic_compute_shape (C++ function), 84
migraphx::op::convolution_backwards::group (C++ member), 85
migraphx::op::convolution_backwards::kdims (C++ function), 85
migraphx::op::convolution_backwards::name (C++ function), 84
migraphx::op::convolution_backwards::padding (C++ member), 85
migraphx::op::convolution_backwards::padding_mode (C++ member), 85
migraphx::op::convolution_backwards::reflect (C++ function), 85
migraphx::op::convolution_backwards::static_compute_shape (C++ function), 84
migraphx::op::convolution_backwards::stride (C++ member), 85
migraphx::op::cos (C++ struct), 85
migraphx::op::cos::apply (C++ function), 85
migraphx::op::cosh (C++ struct), 85
migraphx::op::cosh::apply (C++ function), 85
migraphx::op::dequantizelinear (C++ struct), 85
migraphx::op::dequantizelinear::attributes (C++ function), 85
migraphx::op::dequantizelinear::compute (C++ function), 85
migraphx::op::dequantizelinear::compute_shape (C++ function), 85
migraphx::op::dequantizelinear::name (C++ function), 85
migraphx::op::dimensions_of (C++ struct), 85
migraphx::op::dimensions_of::compute (C++ function), 86
migraphx::op::dimensions_of::compute_shape (C++ function), 86
migraphx::op::dimensions_of::end (C++ member), 86
migraphx::op::dimensions_of::name (C++ function), 86
migraphx::op::dimensions_of::reflect (C++ function), 86
migraphx::op::dimensions_of::start (C++ member), 86
migraphx::op::div (C++ struct), 86
migraphx::op::div::apply (C++ function), 86
migraphx::op::div::point_function (C++ function), 86
migraphx::op::dot (C++ struct), 86
migraphx::op::dot::compute (C++ function), 86
migraphx::op::dot::compute_shape (C++ function), 86
migraphx::op::dot::name (C++ function), 86
migraphx::op::elu (C++ struct), 86
migraphx::op::elu::alpha (C++ member), 87
migraphx::op::elu::apply (C++ function), 87
migraphx::op::elu::point_op (C++ function), 87
migraphx::op::elu::reflect (C++ function), 87
migraphx::op::equal (C++ struct), 87
migraphx::op::equal::apply (C++ function), 87
migraphx::op::equal::attributes (C++ function), 87
migraphx::op::equal::point_function (C++ function), 87
migraphx::op::erf (C++ struct), 87
migraphx::op::erf::apply (C++ function), 87
migraphx::op::exp (C++ struct), 87
migraphx::op::exp::apply (C++ function), 87
migraphx::op::fill (C++ struct), 87
migraphx::op::fill::compute (C++ function), 87
migraphx::op::fill::compute_shape (C++ function), 87
migraphx::op::fill::name (C++ function), 87
migraphx::op::fill::output_alias (C++ function), 87
migraphx::op::fixed_pad (C++ struct), 88
migraphx::op::fixed_pad::compute (C++ function), 88
migraphx::op::fixed_pad::compute_shape (C++ function), 88
migraphx::op::fixed_pad::name (C++ function), 88
migraphx::op::flatten (C++ struct), 88
migraphx::op::flatten::attributes (C++ function), 88
migraphx::op::flatten::axis (C++ member), 88
migraphx::op::flatten::compute (C++ function), 88
migraphx::op::flatten::name (C++ function), 88
migraphx::op::flatten::normalize_compute_shape (C++ function), 88
migraphx::op::flatten::reflect (C++ function), 88
migraphx::op::floor (C++ struct), 88
migraphx::op::floor::apply (C++ function), 88
migraphx::op::fmod (C++ struct), 88
migraphx::op::fmod::apply (C++ function), 88
migraphx::op::fmod::attributes (C++ function), 88
migraphx::op::fmod::name (C++ function), 88
migraphx::op::gather (C++ struct), 89
migraphx::op::gather::attributes (C++ function), 89
migraphx::op::gather::axis (C++ member), 89
migraphx::op::gather::compute (C++ function), 89
migraphx::op::gather::name (C++ function), 89
migraphx::op::gather::normalize_compute_shape (C++ function), 89

migraphx::op::gather::reflect (*C++ function*), 89
migraphx::op::gathernd (*C++ struct*), 89
migraphx::op::gathernd::batch_dims (*C++ member*), 89
migraphx::op::gathernd::compute (*C++ function*), 89
migraphx::op::gathernd::compute_shape (*C++ function*), 89
migraphx::op::gathernd::name (*C++ function*), 89
migraphx::op::gathernd::reflect (*C++ function*), 89
migraphx::op::get_tuple_elem (*C++ struct*), 89
migraphx::op::get_tuple_elem::compute (*C++ function*), 90
migraphx::op::get_tuple_elem::compute_shape (*C++ function*), 90
migraphx::op::get_tuple_elem::index (*C++ member*), 90
migraphx::op::get_tuple_elem::name (*C++ function*), 90
migraphx::op::get_tuple_elem::output_alias (*C++ function*), 90
migraphx::op::get_tuple_elem::reflect (*C++ function*), 90
migraphx::op::gqa_rotary_embedding (*C++ struct*), 90
migraphx::op::gqa_rotary_embedding::compute (*C++ function*), 90
migraphx::op::gqa_rotary_embedding::compute_shape (*C++ function*), 90
migraphx::op::gqa_rotary_embedding::interleaved (*C++ member*), 90
migraphx::op::gqa_rotary_embedding::kv_num_heads (*C++ member*), 90
migraphx::op::gqa_rotary_embedding::name (*C++ function*), 90
migraphx::op::gqa_rotary_embedding::num_heads (*C++ member*), 90
migraphx::op::gqa_rotary_embedding::pack_v_into_rotary_qkv (*C++ function*), 90
migraphx::op::gqa_rotary_embedding::reflect (*C++ function*), 90
migraphx::op::gqa_rotary_embedding::rotary_parameters (*C++ struct*), 90
migraphx::op::gqa_rotary_embedding::rotary_parameters::batch_size (*C++ member*), 91
migraphx::op::gqa_rotary_embedding::rotary_parameters::batch_stride (*C++ member*), 91
migraphx::op::gqa_rotary_embedding::rotary_parameters::head_size (*C++ member*), 91
migraphx::op::gqa_rotary_embedding::rotary_parameters::head_stride (*C++ member*), 91
migraphx::op::gqa_rotary_embedding::rotary_parameters::max_sequence_length (*C++ member*), 91
migraphx::op::gqa_rotary_embedding::rotary_parameters::num (*C++ member*), 91
migraphx::op::gqa_rotary_embedding::rotary_parameters::pos (*C++ member*), 91
migraphx::op::gqa_rotary_embedding::rotary_parameters::rot (*C++ member*), 91
migraphx::op::gqa_rotary_embedding::rotary_parameters::seq (*C++ member*), 91
migraphx::op::gqa_rotary_embedding::rotary_parameters::seq (*C++ member*), 91
migraphx::op::gqa_rotary_embedding::run_rotary_embedding (*C++ function*), 90
migraphx::op::greater (*C++ struct*), 91
migraphx::op::greater::apply (*C++ function*), 91
migraphx::op::greater::point_function (*C++ function*), 91
migraphx::op::group (*C++ struct*), 91
migraphx::op::group::compute_shape (*C++ function*), 91
migraphx::op::group::name (*C++ function*), 91
migraphx::op::group::reflect (*C++ function*), 92
migraphx::op::group::tag (*C++ member*), 91
migraphx::op::gru (*C++ struct*), 92
migraphx::op::gru::actv_funcs (*C++ member*), 92
migraphx::op::gru::clip (*C++ member*), 92
migraphx::op::gru::compute_shape (*C++ function*), 92
migraphx::op::gru::direction (*C++ member*), 92
migraphx::op::gru::hidden_size (*C++ member*), 92
migraphx::op::gru::linear_before_reset (*C++ member*), 92
migraphx::op::gru::name (*C++ function*), 92
migraphx::op::gru::reflect (*C++ function*), 92
migraphx::op::highest (*C++ struct*), 92
migraphx::op::highest::operator T (*C++ function*), 92
migraphx::op::identity (*C++ struct*), 92
migraphx::op::identity::attributes (*C++ function*), 92
migraphx::op::identity::compute (*C++ function*), 92
migraphx::op::identity::compute_shape (*C++ function*), 92
migraphx::op::identity::name (*C++ function*), 92
migraphx::op::identity::output_alias (*C++ function*), 93
migraphx::op::if_op (*C++ struct*), 93
migraphx::op::if_op::compute (*C++ function*), 93
migraphx::op::if_op::compute_shape (*C++ function*), 93
migraphx::op::if_op::name (*C++ function*), 93
migraphx::op::im2col (*C++ struct*), 93

migraphx::op::im2col::attributes (*C++ function*), 93
migraphx::op::im2col::dilation (*C++ member*), 93
migraphx::op::im2col::name (*C++ function*), 93
migraphx::op::im2col::normalize_compute_shape (*C++ function*), 93
migraphx::op::im2col::padding (*C++ member*), 93
migraphx::op::im2col::padding_mode (*C++ member*), 93
migraphx::op::im2col::reflect (*C++ function*), 93
migraphx::op::im2col::stride (*C++ member*), 93
migraphx::op::isinf (*C++ struct*), 93
migraphx::op::isinf::apply (*C++ function*), 94
migraphx::op::isinf::compute_shape (*C++ function*), 94
migraphx::op::isinf::name (*C++ function*), 94
migraphx::op::isnan (*C++ struct*), 94
migraphx::op::isnan::apply (*C++ function*), 94
migraphx::op::isnan::compute_shape (*C++ function*), 94
migraphx::op::isnan::name (*C++ function*), 94
migraphx::op::layout (*C++ struct*), 94
migraphx::op::layout::apply (*C++ function*), 94
migraphx::op::layout::compute_shape (*C++ function*), 94
migraphx::op::layout::permutation (*C++ member*), 94
migraphx::op::layout::reflect (*C++ function*), 94
migraphx::op::leaky_relu (*C++ struct*), 94
migraphx::op::leaky_relu::alpha (*C++ member*), 95
migraphx::op::leaky_relu::apply (*C++ function*), 94
migraphx::op::leaky_relu::name (*C++ function*), 94
migraphx::op::leaky_relu::point_op (*C++ function*), 94
migraphx::op::leaky_relu::reflect (*C++ function*), 95
migraphx::op::less (*C++ struct*), 95
migraphx::op::less::apply (*C++ function*), 95
migraphx::op::less::point_function (*C++ function*), 95
migraphx::op::load (*C++ struct*), 95
migraphx::op::load::compute (*C++ function*), 95
migraphx::op::load::compute_shape (*C++ function*), 95
migraphx::op::load::get_lifetime (*C++ function*), 95
migraphx::op::load::name (*C++ function*), 95
migraphx::op::load::offset (*C++ member*), 95
migraphx::op::load::operator<< (*C++ function*), 95
migraphx::op::load::output_alias (*C++ function*), 95
migraphx::op::load::reflect (*C++ function*), 95
migraphx::op::load::s (*C++ member*), 95
migraphx::op::log (*C++ struct*), 95
migraphx::op::log2 (*C++ struct*), 96
migraphx::op::log2::apply (*C++ function*), 96
migraphx::op::log::apply (*C++ function*), 96
migraphx::op::logical_and (*C++ struct*), 96
migraphx::op::logical_and::apply (*C++ function*), 96
migraphx::op::logical_and::point_function (*C++ function*), 96
migraphx::op::logical_or (*C++ struct*), 96
migraphx::op::logical_or::apply (*C++ function*), 96
migraphx::op::logical_or::point_function (*C++ function*), 96
migraphx::op::logical_xor (*C++ struct*), 96
migraphx::op::logical_xor::apply (*C++ function*), 96
migraphx::op::logical_xor::point_function (*C++ function*), 96
migraphx::op::logsoftmax (*C++ struct*), 96
migraphx::op::logsoftmax::attributes (*C++ function*), 96
migraphx::op::logsoftmax::axis (*C++ member*), 97
migraphx::op::logsoftmax::name (*C++ function*), 96
migraphx::op::logsoftmax::normalize_compute_shape (*C++ function*), 96
migraphx::op::logsoftmax::output (*C++ function*), 96
migraphx::op::logsoftmax::reflect (*C++ function*), 97
migraphx::op::loop (*C++ struct*), 97
migraphx::op::loop::compute (*C++ function*), 97
migraphx::op::loop::compute_shape (*C++ function*), 97
migraphx::op::loop::max_iterations (*C++ member*), 97
migraphx::op::loop::name (*C++ function*), 97
migraphx::op::loop::ref_loop (*C++ struct*), 97
migraphx::op::loop::ref_loop::append (*C++ function*), 97
migraphx::op::loop::ref_loop::copy (*C++ function*), 97
migraphx::op::loop::ref_loop::get_output_params (*C++ function*), 97
migraphx::op::loop::ref_loop::max_iterations (*C++ member*), 98
migraphx::op::loop::ref_loop::set_zero (*C++ function*), 97

migraphx::op::loop::reflect (*C++ function*), 97
migraphx::op::loop::scan_output_directions (*C++ member*), 97
migraphx::op::lowest (*C++ struct*), 98
migraphx::op::lowest::operator T (*C++ function*), 98
migraphx::op::lrn (*C++ struct*), 98
migraphx::op::lrn::alpha (*C++ member*), 98
migraphx::op::lrn::beta (*C++ member*), 98
migraphx::op::lrn::bias (*C++ member*), 98
migraphx::op::lrn::compute_shape (*C++ function*), 98
migraphx::op::lrn::name (*C++ function*), 98
migraphx::op::lrn::reflect (*C++ function*), 98
migraphx::op::lrn::size (*C++ member*), 98
migraphx::op::lstm (*C++ struct*), 98
migraphx::op::lstm::actv_funcs (*C++ member*), 98
migraphx::op::lstm::clip (*C++ member*), 99
migraphx::op::lstm::compute_shape (*C++ function*), 98
migraphx::op::lstm::direction (*C++ member*), 99
migraphx::op::lstm::hidden_size (*C++ member*), 98
migraphx::op::lstm::input_forget (*C++ member*), 99
migraphx::op::lstm::name (*C++ function*), 98
migraphx::op::lstm::reflect (*C++ function*), 99
migraphx::op::max (*C++ struct*), 99
migraphx::op::max::apply (*C++ function*), 99
migraphx::op::max::attributes (*C++ function*), 99
migraphx::op::min (*C++ struct*), 99
migraphx::op::min::apply (*C++ function*), 99
migraphx::op::min::attributes (*C++ function*), 99
migraphx::op::mod (*C++ struct*), 99
migraphx::op::mod::apply (*C++ function*), 99
migraphx::op::mod::attributes (*C++ function*), 99
migraphx::op::mod::name (*C++ function*), 99
migraphx::op::mul (*C++ struct*), 99
migraphx::op::mul::apply (*C++ function*), 99
migraphx::op::mul::attributes (*C++ function*), 99
migraphx::op::mul::point_function (*C++ function*), 99
migraphx::op::multibroadcast (*C++ struct*), 100
migraphx::op::multibroadcast::compute (*C++ function*), 100
migraphx::op::multibroadcast::compute_shape (*C++ function*), 100
migraphx::op::multibroadcast::name (*C++ function*), 100
migraphx::op::multibroadcast::output_alias (*C++ function*), 100
migraphx::op::multibroadcast::output_dyn_dims (*C++ member*), 100
migraphx::op::multibroadcast::output_lens (*C++ member*), 100
migraphx::op::multibroadcast::reflect (*C++ function*), 100
migraphx::op::multinomial (*C++ struct*), 100
migraphx::op::multinomial::compute (*C++ function*), 100
migraphx::op::multinomial::compute_shape (*C++ function*), 100
migraphx::op::multinomial::dtype (*C++ member*), 100
migraphx::op::multinomial::name (*C++ function*), 100
migraphx::op::multinomial::reflect (*C++ function*), 100
migraphx::op::nearbyint (*C++ struct*), 100
migraphx::op::nearbyint::apply (*C++ function*), 101
migraphx::op::neg (*C++ struct*), 101
migraphx::op::neg::apply (*C++ function*), 101
migraphx::op::neg::point_function (*C++ function*), 101
migraphx::op::nonmaxsuppression (*C++ struct*), 101
migraphx::op::nonmaxsuppression::batch_box (*C++ function*), 101
migraphx::op::nonmaxsuppression::box (*C++ struct*), 101
migraphx::op::nonmaxsuppression::box::area (*C++ function*), 102
migraphx::op::nonmaxsuppression::box::operator[] (*C++ function*), 102
migraphx::op::nonmaxsuppression::box::sort (*C++ function*), 102
migraphx::op::nonmaxsuppression::box::x (*C++ member*), 102
migraphx::op::nonmaxsuppression::box::y (*C++ member*), 102
migraphx::op::nonmaxsuppression::center_point_box (*C++ member*), 101
migraphx::op::nonmaxsuppression::compute (*C++ function*), 101
migraphx::op::nonmaxsuppression::compute_nms (*C++ function*), 101
migraphx::op::nonmaxsuppression::compute_shape (*C++ function*), 101
migraphx::op::nonmaxsuppression::filter_boxes_by_score (*C++ function*), 101
migraphx::op::nonmaxsuppression::name (*C++ function*), 101
migraphx::op::nonmaxsuppression::reflect (*C++ function*), 101
migraphx::op::nonmaxsuppression::suppress_by_iou (*C++ function*), 101

migraphx::op::nonmaxsuppression::use_dyn_output (*C++ member*), 101
migraphx::op::nonzero (*C++ struct*), 102
migraphx::op::nonzero::compute (*C++ function*), 102
migraphx::op::nonzero::compute_shape (*C++ function*), 102
migraphx::op::nonzero::name (*C++ function*), 102
migraphx::op::normalize_attribute (*C++ enum*), 74
migraphx::op::normalize_attribute::clip_max (*C++ enumerator*), 74
migraphx::op::normalize_attribute::clip_min (*C++ enumerator*), 74
migraphx::op::normalize_attribute::include_max (*C++ enumerator*), 75
migraphx::op::normalize_attribute::include_min (*C++ enumerator*), 75
migraphx::op::normalize_attribute::normalize_padding (*C++ enumerator*), 75
migraphx::op::normalize_attribute::use_len (*C++ enumerator*), 74
migraphx::op::normalize_attribute::use_output (*C++ enumerator*), 74
migraphx::op::one (*C++ struct*), 102
migraphx::op::one::operator T (*C++ function*), 102
migraphx::op::onehot (*C++ struct*), 102
migraphx::op::onehot::axis (*C++ member*), 103
migraphx::op::onehot::compute (*C++ function*), 102
migraphx::op::onehot::compute_shape (*C++ function*), 102
migraphx::op::onehot::depth (*C++ member*), 103
migraphx::op::onehot::name (*C++ function*), 102
migraphx::op::onehot::reflect (*C++ function*), 103
migraphx::op::op_name (*C++ struct*), 103
migraphx::op::op_name::name (*C++ function*), 104
migraphx::op::operator<< (*C++ function*), 75
migraphx::op::outline (*C++ struct*), 104
migraphx::op::outline::compute (*C++ function*), 104
migraphx::op::outline::compute_shape (*C++ function*), 104
migraphx::op::outline::name (*C++ function*), 104
migraphx::op::outline::reflect (*C++ function*), 104
migraphx::op::outline::s (*C++ member*), 104
migraphx::op::pack_fp4 (*C++ struct*), 104
migraphx::op::pack_fp4::compute (*C++ function*), 104
migraphx::op::pack_fp4::name (*C++ function*), 104
migraphx::op::pack_fp4::normalize_compute_shape (*C++ function*), 104
migraphx::op::pack_int4 (*C++ struct*), 104
migraphx::op::pack_int4::attributes (*C++ function*), 104
migraphx::op::pack_int4::axis (*C++ member*), 105
migraphx::op::pack_int4::compute (*C++ function*), 104
migraphx::op::pack_int4::name (*C++ function*), 104
migraphx::op::pack_int4::normalize_compute_shape (*C++ function*), 104
migraphx::op::pack_int4::reflect (*C++ function*), 105
migraphx::op::pad (*C++ struct*), 105
migraphx::op::pad::compute_shape (*C++ function*), 105
migraphx::op::pad::mode (*C++ member*), 105
migraphx::op::pad::name (*C++ function*), 105
migraphx::op::pad::pad_ndims (*C++ function*), 105
migraphx::op::pad::pad_op_mode_t (*C++ enum*), 105
migraphx::op::pad::pad_op_mode_t::constant_pad (*C++ enumerator*), 105
migraphx::op::pad::pad_op_mode_t::edge_pad (*C++ enumerator*), 105
migraphx::op::pad::pad_op_mode_t::reflect_pad (*C++ enumerator*), 105
migraphx::op::pad::pads (*C++ member*), 105
migraphx::op::pad::reflect (*C++ function*), 105
migraphx::op::pad::symmetric (*C++ function*), 105
migraphx::op::pad::value (*C++ member*), 105
migraphx::op::padding_mode_t (*C++ enum*), 73
migraphx::op::padding_mode_t::default_ (*C++ enumerator*), 73
migraphx::op::padding_mode_t::same_lower (*C++ enumerator*), 73
migraphx::op::padding_mode_t::same_upper (*C++ enumerator*), 73
migraphx::op::pointwise (*C++ struct*), 105
migraphx::op::pointwise::attributes (*C++ function*), 106
migraphx::op::pointwise::compute (*C++ function*), 106
migraphx::op::pointwise::compute_shape (*C++ function*), 106
migraphx::op::pointwise::name (*C++ function*), 106
migraphx::op::pointwise::remove_broadcasts (*C++ function*), 106
migraphx::op::pooling (*C++ struct*), 106
migraphx::op::pooling::attributes (*C++ function*), 106
migraphx::op::pooling::avg_pool (*C++ struct*),

107
migraphx::op::pooling::avg_pool::final (*C++ function*), 107
migraphx::op::pooling::avg_pool::init (*C++ function*), 107
migraphx::op::pooling::avg_pool::operator() (*C++ function*), 107
migraphx::op::pooling::calc_pooling (*C++ function*), 106
migraphx::op::pooling::calc_spatial_dim_out (*C++ function*), 106
migraphx::op::pooling::ceil_mode (*C++ member*), 107
migraphx::op::pooling::check_attribute_size (*C++ function*), 106
migraphx::op::pooling::compute (*C++ function*), 106
migraphx::op::pooling::count_include_pad (*C++ member*), 107
migraphx::op::pooling::dilate_dim (*C++ function*), 106
migraphx::op::pooling::dilations (*C++ member*), 106
migraphx::op::pooling::dyn_global (*C++ member*), 107
migraphx::op::pooling::kdims (*C++ function*), 106
migraphx::op::pooling::lengths (*C++ member*), 106
migraphx::op::pooling::lp_order (*C++ member*), 107
migraphx::op::pooling::lpnorm_pool (*C++ struct*), 107
migraphx::op::pooling::lpnorm_pool::final (*C++ function*), 107
migraphx::op::pooling::lpnorm_pool::init (*C++ function*), 107
migraphx::op::pooling::lpnorm_pool::lpnorm_pool (*C++ function*), 107
migraphx::op::pooling::lpnorm_pool::operator() (*C++ function*), 107
migraphx::op::pooling::lpnorm_pool::p (*C++ member*), 107
migraphx::op::pooling::max_pool (*C++ struct*), 107
migraphx::op::pooling::max_pool::final (*C++ function*), 108
migraphx::op::pooling::max_pool::init (*C++ function*), 108
migraphx::op::pooling::max_pool::operator() (*C++ function*), 108
migraphx::op::pooling::mode (*C++ member*), 106
migraphx::op::pooling::name (*C++ function*), 106
migraphx::op::pooling::normalize_compute_shape (*C++ function*), 106
migraphx::op::pooling::padding (*C++ member*), 106
migraphx::op::pooling::padding_mode (*C++ member*), 107
migraphx::op::pooling::reflect (*C++ function*), 107
migraphx::op::pooling::stride (*C++ member*), 106
migraphx::op::pooling_mode (*C++ enum*), 74
migraphx::op::pooling_mode::average (*C++ enumerator*), 74
migraphx::op::pooling_mode::lpnorm (*C++ enumerator*), 74
migraphx::op::pooling_mode::max (*C++ enumerator*), 74
migraphx::op::pow (*C++ struct*), 108
migraphx::op::pow::apply (*C++ function*), 108
migraphx::op::prefix_scan_op (*C++ struct*), 108
migraphx::op::prefix_scan_op::attributes (*C++ function*), 108
migraphx::op::prefix_scan_op::axis (*C++ member*), 108
migraphx::op::prefix_scan_op::compute (*C++ function*), 108
migraphx::op::prefix_scan_op::exclusive (*C++ member*), 108
migraphx::op::prefix_scan_op::init (*C++ function*), 108
migraphx::op::prefix_scan_op::normalize_compute_shape (*C++ function*), 108
migraphx::op::prefix_scan_op::prefix_scan_op (*C++ function*), 108
migraphx::op::prefix_scan_op::reflect (*C++ function*), 109
migraphx::op::prefix_scan_op::reverse (*C++ member*), 108
migraphx::op::prefix_scan_sum (*C++ struct*), 109
migraphx::op::prefix_scan_sum::op (*C++ function*), 109
migraphx::op::prefix_scan_sum::prefix_scan_sum (*C++ function*), 109
migraphx::op::prelu (*C++ struct*), 109
migraphx::op::prelu::apply (*C++ function*), 109
migraphx::op::prelu::point_op (*C++ function*), 109
migraphx::op::quant_convolution (*C++ struct*), 109
migraphx::op::quant_convolution::attributes (*C++ function*), 109
migraphx::op::quant_convolution::check_attribute_size (*C++ function*), 109
migraphx::op::quant_convolution::compute (*C++ function*), 109
migraphx::op::quant_convolution::dilation

(C++ member), 110
migraphx::op::quant_convolution::group (C++ member), 110
migraphx::op::quant_convolution::kdims (C++ function), 109
migraphx::op::quant_convolution::name (C++ function), 109
migraphx::op::quant_convolution::normalize_compute_shape (C++ function), 109
migraphx::op::quant_convolution::padding (C++ member), 109
migraphx::op::quant_convolution::padding_mode (C++ member), 110
migraphx::op::quant_convolution::reflect (C++ function), 110
migraphx::op::quant_convolution::stride (C++ member), 109
migraphx::op::quant_dot (C++ struct), 110
migraphx::op::quant_dot::attributes (C++ function), 110
migraphx::op::quant_dot::compute_shape (C++ function), 110
migraphx::op::quant_dot::name (C++ function), 110
migraphx::op::quantizelinear (C++ struct), 110
migraphx::op::quantizelinear::attributes (C++ function), 110
migraphx::op::quantizelinear::compute (C++ function), 110
migraphx::op::quantizelinear::compute_shape (C++ function), 110
migraphx::op::quantizelinear::name (C++ function), 110
migraphx::op::quantizelinear::out_type (C++ member), 110
migraphx::op::quantizelinear::reflect (C++ function), 110
migraphx::op::random_seed (C++ struct), 110
migraphx::op::random_seed::compute (C++ function), 111
migraphx::op::random_seed::compute_shape (C++ function), 111
migraphx::op::random_seed::dtype (C++ member), 111
migraphx::op::random_seed::name (C++ function), 111
migraphx::op::random_seed::reflect (C++ function), 111
migraphx::op::random_uniform (C++ struct), 111
migraphx::op::random_uniform::compute (C++ function), 111
migraphx::op::random_uniform::compute_shape (C++ function), 111
migraphx::op::random_uniform::name (C++ function), 111
migraphx::op::random_uniform::output_alias (C++ function), 111
migraphx::op::recip (C++ struct), 111
migraphx::op::recip::apply (C++ function), 111
migraphx::op::recip::point_op (C++ function), 111
migraphx::op::reduce_all (C++ struct), 111
migraphx::op::reduce_all::init (C++ function), 112
migraphx::op::reduce_all::op (C++ function), 112
migraphx::op::reduce_all::reduce_all (C++ function), 112
migraphx::op::reduce_any (C++ struct), 112
migraphx::op::reduce_any::op (C++ function), 112
migraphx::op::reduce_any::reduce_any (C++ function), 112
migraphx::op::reduce_max (C++ struct), 112
migraphx::op::reduce_max::init (C++ function), 112
migraphx::op::reduce_max::op (C++ function), 112
migraphx::op::reduce_max::reduce_max (C++ function), 112
migraphx::op::reduce_mean (C++ struct), 112
migraphx::op::reduce_mean::op (C++ function), 112
migraphx::op::reduce_mean::output (C++ function), 112
migraphx::op::reduce_mean::reduce_mean (C++ function), 112
migraphx::op::reduce_min (C++ struct), 112
migraphx::op::reduce_min::init (C++ function), 113
migraphx::op::reduce_min::op (C++ function), 113
migraphx::op::reduce_min::reduce_min (C++ function), 113
migraphx::op::reduce_op (C++ struct), 113
migraphx::op::reduce_op::attributes (C++ function), 113
migraphx::op::reduce_op::axes (C++ member), 114
migraphx::op::reduce_op::collapse_reduced_axes (C++ function), 113
migraphx::op::reduce_op::compute (C++ function), 113
migraphx::op::reduce_op::compute_dynamic_shape (C++ function), 113
migraphx::op::reduce_op::compute_static_shape (C++ function), 113
migraphx::op::reduce_op::init (C++ function), 113
migraphx::op::reduce_op::input (C++ function), 113
migraphx::op::reduce_op::normalize_compute_shape

(C++ function), 113
migraphx::op::reduce_op::output (C++ function), 113
migraphx::op::reduce_op::reduce (C++ function), 113
migraphx::op::reduce_op::reduce_op (C++ function), 113
migraphx::op::reduce_op::reflect (C++ function), 114
migraphx::op::reduce_op::tune_dims (C++ function), 113
migraphx::op::reduce_prod (C++ struct), 114
migraphx::op::reduce_prod::init (C++ function), 114
migraphx::op::reduce_prod::op (C++ function), 114
migraphx::op::reduce_prod::reduce_prod (C++ function), 114
migraphx::op::reduce_sum (C++ struct), 114
migraphx::op::reduce_sum::op (C++ function), 114
migraphx::op::reduce_sum::reduce_sum (C++ function), 114
migraphx::op::relu (C++ struct), 114
migraphx::op::relu::apply (C++ function), 114
migraphx::op::relu::point_op (C++ function), 114
migraphx::op::reshape (C++ struct), 114
migraphx::op::reshape::compute (C++ function), 115
migraphx::op::reshape::compute_shape (C++ function), 115
migraphx::op::reshape::dims (C++ member), 115
migraphx::op::reshape::dyn_1arg_compute_shape (C++ function), 115
migraphx::op::reshape::name (C++ function), 115
migraphx::op::reshape::reflect (C++ function), 115
migraphx::op::reshape::static_compute_shape (C++ function), 115
migraphx::op::reshape_lazy (C++ struct), 115
migraphx::op::reshape_lazy::attributes (C++ function), 115
migraphx::op::reshape_lazy::can_strides_merge (C++ function), 116
migraphx::op::reshape_lazy::compute (C++ function), 115
migraphx::op::reshape_lazy::compute_end_dim (C++ function), 115
migraphx::op::reshape_lazy::compute_shape (C++ function), 115
migraphx::op::reshape_lazy::dims (C++ member), 115
migraphx::op::reshape_lazy::dyn_compute_shape (C++ function), 115
migraphx::op::reshape_lazy::merge_strides (C++ function), 115
migraphx::op::reshape_lazy::name (C++ function), 115
migraphx::op::reshape_lazy::output_alias (C++ function), 115
migraphx::op::reshape_lazy::reflect (C++ function), 115
migraphx::op::reshape_lazy::reshape_lazy_dims (C++ function), 116
migraphx::op::reshape_lazy::static_compute_shape (C++ function), 115
migraphx::op::reshape_lazy::try_merge_pairs (C++ function), 115
migraphx::op::resize (C++ struct), 116
migraphx::op::resize::compute (C++ function), 116
migraphx::op::resize::compute_shape (C++ function), 116
migraphx::op::resize::coordinate_transformation_mode (C++ member), 116
migraphx::op::resize::get_nearest_op (C++ function), 116
migraphx::op::resize::get_original_idx_op (C++ function), 116
migraphx::op::resize::mode (C++ member), 116
migraphx::op::resize::name (C++ function), 116
migraphx::op::resize::nearest_mode (C++ member), 116
migraphx::op::resize::reflect (C++ function), 116
migraphx::op::resize::scales (C++ member), 116
migraphx::op::resize::sizes (C++ member), 116
migraphx::op::reverse (C++ struct), 116
migraphx::op::reverse::attributes (C++ function), 117
migraphx::op::reverse::axes (C++ member), 117
migraphx::op::reverse::compute (C++ function), 117
migraphx::op::reverse::name (C++ function), 117
migraphx::op::reverse::normalize_compute_shape (C++ function), 117
migraphx::op::reverse::reflect (C++ function), 117
migraphx::op::rnn (C++ struct), 117
migraphx::op::rnn::actv_funcs (C++ member), 117
migraphx::op::rnn::clip (C++ member), 117
migraphx::op::rnn::compute_shape (C++ function), 117
migraphx::op::rnn::direction (C++ member), 117
migraphx::op::rnn::hidden_size (C++ member), 117
migraphx::op::rnn::name (C++ function), 117
migraphx::op::rnn::reflect (C++ function), 117

migraphx::op::rnn_direction (C++ enum), 74
migraphx::op::rnn_direction::bidirectional (C++ enumerator), 74
migraphx::op::rnn_direction::forward (C++ enumerator), 74
migraphx::op::rnn_direction::reverse (C++ enumerator), 74
migraphx::op::rnn_last_cell_output (C++ struct), 117
migraphx::op::rnn_last_cell_output::compute_shape (C++ function), 117
migraphx::op::rnn_last_cell_output::name (C++ function), 117
migraphx::op::rnn_last_hs_output (C++ struct), 117
migraphx::op::rnn_last_hs_output::compute_shape (C++ function), 118
migraphx::op::rnn_last_hs_output::name (C++ function), 118
migraphx::op::rnn_var_sl_last_output (C++ struct), 118
migraphx::op::rnn_var_sl_last_output::compute_shape (C++ function), 118
migraphx::op::rnn_var_sl_last_output::direction (C++ member), 118
migraphx::op::rnn_var_sl_last_output::name (C++ function), 118
migraphx::op::rnn_var_sl_last_output::reflect (C++ function), 118
migraphx::op::rnn_var_sl_shift_output (C++ struct), 118
migraphx::op::rnn_var_sl_shift_output::compute (C++ function), 118
migraphx::op::rnn_var_sl_shift_output::compute_shape (C++ function), 118
migraphx::op::rnn_var_sl_shift_output::direction (C++ member), 118
migraphx::op::rnn_var_sl_shift_output::name (C++ function), 118
migraphx::op::rnn_var_sl_shift_output::output_name (C++ member), 118
migraphx::op::rnn_var_sl_shift_output::reflect (C++ function), 118
migraphx::op::rnn_var_sl_shift_sequence (C++ struct), 118
migraphx::op::rnn_var_sl_shift_sequence::compute (C++ function), 119
migraphx::op::rnn_var_sl_shift_sequence::compute_shape (C++ function), 119
migraphx::op::rnn_var_sl_shift_sequence::name (C++ function), 119
migraphx::op::roialign (C++ struct), 119
migraphx::op::roialign::avg_pool (C++ struct), 119
migraphx::op::roialign::avg_pool::final (C++ function), 120
migraphx::op::roialign::avg_pool::init (C++ function), 120
migraphx::op::roialign::avg_pool::operator() (C++ function), 120
migraphx::op::roialign::calc_pooling (C++ function), 119
migraphx::op::roialign::calc_pos_weight (C++ function), 119
migraphx::op::roialign::compute (C++ function), 119
migraphx::op::roialign::compute_shape (C++ function), 119
migraphx::op::roialign::coord_trans_mode (C++ member), 119
migraphx::op::roialign::max_pool (C++ struct), 120
migraphx::op::roialign::max_pool::final (C++ function), 120
migraphx::op::roialign::max_pool::init (C++ function), 120
migraphx::op::roialign::max_pool::operator() (C++ function), 120
migraphx::op::roialign::mode (C++ member), 119
migraphx::op::roialign::name (C++ function), 119
migraphx::op::roialign::output_height (C++ member), 119
migraphx::op::roialign::output_width (C++ member), 119
migraphx::op::roialign::pos_weight (C++ struct), 120
migraphx::op::roialign::pos_weight::pos (C++ member), 120
migraphx::op::roialign::pos_weight::w (C++ member), 120
migraphx::op::roialign::reflect (C++ function), 119
migraphx::op::roialign::sampling_ratio (C++ member), 119
migraphx::op::roialign::spatial_scale (C++ member), 119
migraphx::op::rsqrt (C++ struct), 120
migraphx::op::rsqrt::apply (C++ function), 120
migraphx::op::run_on_target (C++ struct), 120
migraphx::op::run_on_target::compute (C++ function), 120
migraphx::op::run_on_target::compute_shape (C++ function), 120
migraphx::op::run_on_target::name (C++ function), 120
migraphx::op::run_on_target::reflect (C++ function), 121
migraphx::op::run_on_target::target_id (C++

member), 121
migraphx::op::scalar (*C++ struct*), 121
migraphx::op::scalar::compute (*C++ function*), 121
migraphx::op::scalar::compute_shape (*C++ function*), 121
migraphx::op::scalar::name (*C++ function*), 121
migraphx::op::scalar::output_alias (*C++ function*), 121
migraphx::op::scalar::reflect (*C++ function*), 121
migraphx::op::scalar::scalar_bcast_lens (*C++ member*), 121
migraphx::op::scan_slice (*C++ struct*), 121
migraphx::op::scan_slice::attributes (*C++ function*), 121
migraphx::op::scan_slice::axis (*C++ member*), 121
migraphx::op::scan_slice::compute (*C++ function*), 121
migraphx::op::scan_slice::direction (*C++ member*), 121
migraphx::op::scan_slice::normalize_compute_shape (*C++ function*), 121
migraphx::op::scan_slice::reflect (*C++ function*), 122
migraphx::op::scatter_add (*C++ struct*), 122
migraphx::op::scatter_add::reduction (*C++ function*), 122
migraphx::op::scatter_max (*C++ struct*), 122
migraphx::op::scatter_max::reduction (*C++ function*), 122
migraphx::op::scatter_min (*C++ struct*), 122
migraphx::op::scatter_min::reduction (*C++ function*), 122
migraphx::op::scatter_mul (*C++ struct*), 122
migraphx::op::scatter_mul::reduction (*C++ function*), 122
migraphx::op::scatter_none (*C++ struct*), 122
migraphx::op::scatter_none::reduction (*C++ function*), 122
migraphx::op::scatter_op (*C++ struct*), 122
migraphx::op::scatter_op::attributes (*C++ function*), 122
migraphx::op::scatter_op::axis (*C++ member*), 123
migraphx::op::scatter_op::compute (*C++ function*), 123
migraphx::op::scatter_op::derived (*C++ function*), 123
migraphx::op::scatter_op::normalize_compute_shape (*C++ function*), 122
migraphx::op::scatter_op::reflect (*C++ function*), 123
migraphx::op::scatter_op::scatter_reduce_iterate (*C++ function*), 122
migraphx::op::scatter_op::skip_out_of_bounds (*C++ member*), 123
migraphx::op::scatternd_add (*C++ struct*), 123
migraphx::op::scatternd_add::reduction (*C++ function*), 123
migraphx::op::scatternd_add::scatternd_add (*C++ function*), 123
migraphx::op::scatternd_max (*C++ struct*), 123
migraphx::op::scatternd_max::reduction (*C++ function*), 123
migraphx::op::scatternd_max::scatternd_max (*C++ function*), 123
migraphx::op::scatternd_min (*C++ struct*), 123
migraphx::op::scatternd_min::reduction (*C++ function*), 123
migraphx::op::scatternd_min::scatternd_min (*C++ function*), 123
migraphx::op::scatternd_mul (*C++ struct*), 123
migraphx::op::scatternd_mul::reduction (*C++ function*), 124
migraphx::op::scatternd_mul::scatternd_mul (*C++ function*), 124
migraphx::op::scatternd_none (*C++ struct*), 124
migraphx::op::scatternd_none::reduction (*C++ function*), 124
migraphx::op::scatternd_none::scatternd_none (*C++ function*), 124
migraphx::op::scatternd_op (*C++ struct*), 124
migraphx::op::scatternd_op::compute (*C++ function*), 124
migraphx::op::scatternd_op::compute_shape (*C++ function*), 124
migraphx::op::scatternd_op::init (*C++ function*), 124
migraphx::op::scatternd_op::scatternd_op (*C++ function*), 124
migraphx::op::select_module (*C++ struct*), 124
migraphx::op::select_module::compute (*C++ function*), 124
migraphx::op::select_module::compute_shape (*C++ function*), 124
migraphx::op::select_module::get_input_parameter_names (*C++ function*), 124
migraphx::op::select_module::get_output_parameter_names (*C++ function*), 124
migraphx::op::select_module::name (*C++ function*), 124
migraphx::op::select_module::output_alias (*C++ function*), 124
migraphx::op::select_module::output_dyn_shapes (*C++ member*), 125
migraphx::op::select_module::reflect (*C++

function), 125
migraphx::op::sigmoid (*C++ struct*), 125
migraphx::op::sigmoid::apply (*C++ function*), 125
migraphx::op::sigmoid::point_op (*C++ function*), 125
migraphx::op::sign (*C++ struct*), 125
migraphx::op::sign::apply (*C++ function*), 125
migraphx::op::sign::point_op (*C++ function*), 125
migraphx::op::sin (*C++ struct*), 125
migraphx::op::sin::apply (*C++ function*), 125
migraphx::op::sinh (*C++ struct*), 125
migraphx::op::sinh::apply (*C++ function*), 125
migraphx::op::slice (*C++ struct*), 125
migraphx::op::slice::all_set (*C++ member*), 127
migraphx::op::slice::attributes (*C++ function*), 126
migraphx::op::slice::axes (*C++ member*), 127
migraphx::op::slice::axes_only (*C++ member*), 128
migraphx::op::slice::compute (*C++ function*), 127
migraphx::op::slice::compute_offset (*C++ function*), 126
migraphx::op::slice::compute_two_or_more (*C++ function*), 126
migraphx::op::slice::ends (*C++ member*), 127
migraphx::op::slice::ends_axes (*C++ member*), 127
migraphx::op::slice::ends_only (*C++ member*), 128
migraphx::op::slice::get_set_attributes (*C++ function*), 126
migraphx::op::slice::lens_calc (*C++ function*), 126
migraphx::op::slice::name (*C++ function*), 126
migraphx::op::slice::none_set (*C++ member*), 128
migraphx::op::slice::normalize_compute_shape (*C++ function*), 126
migraphx::op::slice::normalize_starts_ends_axes (*C++ function*), 126
migraphx::op::slice::output_alias (*C++ function*), 127
migraphx::op::slice::reflect (*C++ function*), 127
migraphx::op::slice::starts (*C++ member*), 127
migraphx::op::slice::starts_axes (*C++ member*), 127
migraphx::op::slice::starts_ends (*C++ member*), 127
migraphx::op::slice::starts_only (*C++ member*), 128
migraphx::op::softmax (*C++ struct*), 128
migraphx::op::softmax::attributes (*C++ function*), 128
migraphx::op::softmax::axis (*C++ member*), 128
migraphx::op::softmax::name (*C++ function*), 128
migraphx::op::softmax::normalize_compute_shape (*C++ function*), 128
migraphx::op::softmax::output (*C++ function*), 128
migraphx::op::softmax::reflect (*C++ function*), 128
migraphx::op::sqdiff (*C++ struct*), 128
migraphx::op::sqdiff::apply (*C++ function*), 128
migraphx::op::sqdiff::point_op (*C++ function*), 128
migraphx::op::sqrt (*C++ struct*), 128
migraphx::op::sqrt::apply (*C++ function*), 128
migraphx::op::squeeze (*C++ struct*), 128
migraphx::op::squeeze::attributes (*C++ function*), 129
migraphx::op::squeeze::axes (*C++ member*), 129
migraphx::op::squeeze::compute (*C++ function*), 129
migraphx::op::squeeze::name (*C++ function*), 129
migraphx::op::squeeze::normalize_compute_shape (*C++ function*), 129
migraphx::op::squeeze::output_alias (*C++ function*), 129
migraphx::op::squeeze::reflect (*C++ function*), 129
migraphx::op::step (*C++ struct*), 129
migraphx::op::step::attributes (*C++ function*), 129
migraphx::op::step::axes (*C++ member*), 129
migraphx::op::step::compute (*C++ function*), 129
migraphx::op::step::name (*C++ function*), 129
migraphx::op::step::normalize_compute_shape (*C++ function*), 129
migraphx::op::step::output_alias (*C++ function*), 129
migraphx::op::step::reflect (*C++ function*), 129
migraphx::op::step::steps (*C++ member*), 129
migraphx::op::sub (*C++ struct*), 129
migraphx::op::sub::apply (*C++ function*), 130
migraphx::op::sub::point_function (*C++ function*), 130
migraphx::op::tan (*C++ struct*), 130
migraphx::op::tan::apply (*C++ function*), 130
migraphx::op::tanh (*C++ struct*), 130
migraphx::op::tanh::apply (*C++ function*), 130
migraphx::op::topk (*C++ struct*), 130
migraphx::op::topk::attributes (*C++ function*), 130
migraphx::op::topk::axis (*C++ member*), 130
migraphx::op::topk::compare_pair (*C++ function*), 130
migraphx::op::topk::compute (*C++ function*), 130
migraphx::op::topk::k (*C++ member*), 130

migraphx::op::topk::largest (*C++ member*), 130
migraphx::op::topk::name (*C++ function*), 130
migraphx::op::topk::normalize_compute_shape (*C++ function*), 130
migraphx::op::topk::reflect (*C++ function*), 130
migraphx::op::transpose (*C++ struct*), 130
migraphx::op::transpose::compute (*C++ function*), 131
migraphx::op::transpose::compute_shape (*C++ function*), 131
migraphx::op::transpose::dims (*C++ member*), 131
migraphx::op::transpose::name (*C++ function*), 131
migraphx::op::transpose::output_alias (*C++ function*), 131
migraphx::op::transpose::reflect (*C++ function*), 131
migraphx::op::unary (*C++ struct*), 131
migraphx::op::unary::attributes (*C++ function*), 131
migraphx::op::unary::base_attributes (*C++ function*), 131
migraphx::op::unary::compute (*C++ function*), 131
migraphx::op::unary::compute_shape (*C++ function*), 131
migraphx::op::unary::point_function (*C++ function*), 131
migraphx::op::unary::point_op (*C++ function*), 131
migraphx::op::unary_not (*C++ struct*), 131
migraphx::op::unary_not::apply (*C++ function*), 131
migraphx::op::unary_not::name (*C++ function*), 131
migraphx::op::unary_not::point_function (*C++ function*), 131
migraphx::op::undefined (*C++ struct*), 131
migraphx::op::undefined::compute (*C++ function*), 132
migraphx::op::undefined::compute_shape (*C++ function*), 132
migraphx::op::undefined::name (*C++ function*), 132
migraphx::op::unique (*C++ struct*), 132
migraphx::op::unique::axis (*C++ member*), 132
migraphx::op::unique::compute (*C++ function*), 132
migraphx::op::unique::compute_shape (*C++ function*), 132
migraphx::op::unique::make_idx_less_fn (*C++ function*), 132
migraphx::op::unique::name (*C++ function*), 132
migraphx::op::unique::reflect (*C++ function*), 132
migraphx::op::unique::sorted (*C++ member*), 132
migraphx::op::unique::sorted_uniq_indices (*C++ function*), 132
migraphx::op::unique::unsorted_uniq_indices (*C++ function*), 132
migraphx::op::unknown (*C++ struct*), 132
migraphx::op::unknown::compute_shape (*C++ function*), 132
migraphx::op::unknown::name (*C++ function*), 132
migraphx::op::unknown::op (*C++ member*), 132
migraphx::op::unknown::operator<< (*C++ function*), 133
migraphx::op::unknown::reflect (*C++ function*), 133
migraphx::op::unpack_fp4 (*C++ struct*), 133
migraphx::op::unpack_fp4::compute (*C++ function*), 133
migraphx::op::unpack_fp4::name (*C++ function*), 133
migraphx::op::unpack_fp4::normalize_compute_shape (*C++ function*), 133
migraphx::op::unpack_int4 (*C++ struct*), 133
migraphx::op::unpack_int4::attributes (*C++ function*), 133
migraphx::op::unpack_int4::axis (*C++ member*), 133
migraphx::op::unpack_int4::compute (*C++ function*), 133
migraphx::op::unpack_int4::name (*C++ function*), 133
migraphx::op::unpack_int4::normalize_compute_shape (*C++ function*), 133
migraphx::op::unpack_int4::reflect (*C++ function*), 133
migraphx::op::unsqueeze (*C++ struct*), 133
migraphx::op::unsqueeze::attributes (*C++ function*), 134
migraphx::op::unsqueeze::axes (*C++ member*), 134
migraphx::op::unsqueeze::compute (*C++ function*), 134
migraphx::op::unsqueeze::name (*C++ function*), 134
migraphx::op::unsqueeze::normalize_compute_shape (*C++ function*), 134
migraphx::op::unsqueeze::output_alias (*C++ function*), 134
migraphx::op::unsqueeze::reflect (*C++ function*), 134
migraphx::op::unsqueeze::steps (*C++ member*), 134
migraphx::op::where (*C++ struct*), 134
migraphx::op::where::attributes (*C++ function*),

134
migraphx::op::where::compute (C++ function), 134
migraphx::op::where::compute_shape (C++ function), 134
migraphx::op::where::name (C++ function), 134
migraphx::op::zero (C++ struct), 134
migraphx::op::zero::operator T (C++ function), 134
migraphx::parse_onnx (C++ function), 18
migraphx::parse_onnx_buffer (C++ function), 18
migraphx::pass (C++ struct), 149
migraphx::pass::apply (C++ function), 149
migraphx::pass::name (C++ function), 149
migraphx::program (C++ struct), 16
migraphx::program::compile (C++ function), 16
migraphx::program::eval (C++ function), 16
migraphx::program::experimental_get_context (C++ function), 16
migraphx::program::get_output_shapes (C++ function), 16
migraphx::program::get_parameter_shapes (C++ function), 16
migraphx::program::migraphx_program_get_main_module (C++ function), 16
migraphx::program::operator!= (C++ function), 17
migraphx::program::operator== (C++ function), 17
migraphx::program::print (C++ function), 16
migraphx::program::program (C++ function), 16
migraphx::program::run_async (C++ function), 16
migraphx::program::sort (C++ function), 16
migraphx::program_parameter_shapes (C++ struct), 15
migraphx::program_parameter_shapes::names (C++ function), 15
migraphx::program_parameter_shapes::operator[] (C++ function), 15
migraphx::program_parameter_shapes::program_parameter_shapes (C++ function), 15
migraphx::program_parameter_shapes::size (C++ function), 15
migraphx::program_parameters (C++ struct), 15
migraphx::program_parameters::add (C++ function), 15
migraphx::program_parameters::program_parameters (C++ function), 15
migraphx::propagate_constant (C++ struct), 151
migraphx::propagate_constant::apply (C++ function), 151
migraphx::propagate_constant::name (C++ function), 151
migraphx::propagate_constant::skip_ops (C++ member), 151
migraphx::quantize_bf16 (C++ function), 17
migraphx::quantize_fp16 (C++ function), 17
migraphx::quantize_int8_options (C++ struct), 17
migraphx::quantize_int8_options::add_calibration_data (C++ function), 17
migraphx::quantize_int8_options::add_op_name (C++ function), 17
migraphx::quantize_int8_options::calibration (C++ member), 17
migraphx::quantize_int8_options::op_names (C++ member), 17
migraphx::quantize_int8_options::quantize_int8_options (C++ function), 17
migraphx::quantize_op_names (C++ struct), 17
migraphx::quantize_op_names::add (C++ function), 17
migraphx::quantize_op_names::quantize_op_names (C++ function), 17
migraphx::raw_data (C++ struct), 68
migraphx::raw_data::at (C++ function), 69
migraphx::raw_data::auto_cast (C++ struct), 70
migraphx::raw_data::auto_cast::get_data_type (C++ type), 70
migraphx::raw_data::auto_cast::is_data_ptr (C++ type), 70
migraphx::raw_data::auto_cast::matches (C++ function), 70
migraphx::raw_data::auto_cast::operator T (C++ function), 70
migraphx::raw_data::auto_cast::operator T* (C++ function), 70
migraphx::raw_data::auto_cast::self (C++ member), 70
migraphx::raw_data::cast (C++ function), 69
migraphx::raw_data::fallback_visit (C++ function), 69
migraphx::raw_data::get (C++ function), 69
migraphx::raw_data::implicit (C++ function), 69
migraphx::raw_data::operator<< (C++ function), 70
migraphx::raw_data::single (C++ function), 69
migraphx::raw_data::to_string (C++ function), 69
migraphx::raw_data::to_vector (C++ function), 69
migraphx::raw_data::visit (C++ function), 69
migraphx::raw_data::visit_at (C++ function), 68
migraphx::rewrite_rnn (C++ struct), 151
migraphx::rewrite_rnn::apply (C++ function), 151
migraphx::rewrite_rnn::name (C++ function), 151
migraphx::save (C++ function), 19
migraphx::schedule (C++ struct), 151
migraphx::schedule::apply (C++ function), 151
migraphx::schedule::enable (C++ member), 151
migraphx::schedule::model (C++ member), 151
migraphx::schedule::name (C++ function), 151
migraphx::shape (C++ struct), 12

migraphx::shape::broadcasted (*C++ function*), 13
migraphx::shape::bytes (*C++ function*), 13
migraphx::shape::compute_index (*C++ function*), 13
migraphx::shape::dyn_dims (*C++ function*), 13
migraphx::shape::dynamic (*C++ function*), 13
migraphx::shape::element_space (*C++ function*), 13
migraphx::shape::elements (*C++ function*), 13
migraphx::shape::get_shape (*C++ function*), 13
migraphx::shape::index (*C++ function*), 13
migraphx::shape::index_array (*C++ type*), 12
migraphx::shape::lengths (*C++ function*), 13
migraphx::shape::lens (*C++ member*), 13
migraphx::shape::multi (*C++ function*), 13
migraphx::shape::operator!= (*C++ function*), 14
migraphx::shape::operator== (*C++ function*), 14
migraphx::shape::operator<< (*C++ function*), 14
migraphx::shape::packed (*C++ function*), 13
migraphx::shape::shape (*C++ function*), 12, 13
migraphx::shape::shape_type (*C++ type*), 12
migraphx::shape::single (*C++ function*), 13
migraphx::shape::skips (*C++ function*), 13
migraphx::shape::standard (*C++ function*), 13
migraphx::shape::strides (*C++ function*), 13
migraphx::shape::strides (*C++ member*), 13
migraphx::shape::transposed (*C++ function*), 13
migraphx::shape::type (*C++ function*), 13
migraphx::simplify_algebra (*C++ struct*), 151
migraphx::simplify_algebra::apply (*C++ function*), 152
migraphx::simplify_algebra::name (*C++ function*), 152
migraphx::simplify_reshapes (*C++ struct*), 152
migraphx::simplify_reshapes::apply (*C++ function*), 152
migraphx::simplify_reshapes::depth (*C++ member*), 152
migraphx::simplify_reshapes::enable_gather_rewrite (*C++ member*), 152
migraphx::simplify_reshapes::enable_op_shape_transform_op (*C++ member*), 152
migraphx::simplify_reshapes::name (*C++ function*), 152
migraphx::target (*C++ struct*), 15
migraphx::target::target (*C++ function*), 15
migraphx_compile_options (*C++ struct*), 15
migraphx_compile_options::migraphx_compile_options (*C++ function*), 16
migraphx_compile_options::object (*C++ member*), 16
migraphx_shape_datatype_t (*C++ enum*), 11
migraphx_shape_datatype_t::migraphx_shape_bf16_type (*C++ enumerator*), 12
migraphx_shape_datatype_t::migraphx_shape_bool_type (*C++ enumerator*), 11
migraphx_shape_datatype_t::migraphx_shape_double_type (*C++ enumerator*), 11
migraphx_shape_datatype_t::migraphx_shape_float_type (*C++ enumerator*), 11
migraphx_shape_datatype_t::migraphx_shape_fp4x2_type (*C++ enumerator*), 11
migraphx_shape_datatype_t::migraphx_shape_fp8e4m3fn_type (*C++ enumerator*), 12
migraphx_shape_datatype_t::migraphx_shape_fp8e4m3fnuz_type (*C++ enumerator*), 12
migraphx_shape_datatype_t::migraphx_shape_fp8e5m2_type (*C++ enumerator*), 12
migraphx_shape_datatype_t::migraphx_shape_fp8e5m2fnuz_type (*C++ enumerator*), 12
migraphx_shape_datatype_t::migraphx_shape_half_type (*C++ enumerator*), 11
migraphx_shape_datatype_t::migraphx_shape_int16_type (*C++ enumerator*), 11
migraphx_shape_datatype_t::migraphx_shape_int32_type (*C++ enumerator*), 11
migraphx_shape_datatype_t::migraphx_shape_int64_type (*C++ enumerator*), 11
migraphx_shape_datatype_t::migraphx_shape_int8_type (*C++ enumerator*), 11
migraphx_shape_datatype_t::migraphx_shape_tuple_type (*C++ enumerator*), 11
migraphx_shape_datatype_t::migraphx_shape_uint16_type (*C++ enumerator*), 11
migraphx_shape_datatype_t::migraphx_shape_uint32_type (*C++ enumerator*), 12
migraphx_shape_datatype_t::migraphx_shape_uint64_type (*C++ enumerator*), 12
migraphx_shape_datatype_t::migraphx_shape_uint8_type (*C++ enumerator*), 11
module
    migraphx, 19

N

ndim() (*in module migraphx*), 20

P

packed() (*in module migraphx*), 20
parse_onnx() (*in module migraphx*), 25
parse_tf() (*in module migraphx*), 26
print() (*in module migraphx*), 23
program (*class in migraphx*), 23

Q

quantize_bf16() (*in module migraphx*), 24
quantize_fp16() (*in module migraphx*), 24
quantize_int8() (*in module migraphx*), 25

R

`run()` (*in module migraphx*), 24

S

`save()` (*in module migraphx*), 26
`save()` (*migraphx.argument static method*), 22
`scalar()` (*in module migraphx*), 20
`shape` (*class in migraphx*), 19
`sort()` (*in module migraphx*), 24
`standard()` (*in module migraphx*), 20
`strides()` (*in module migraphx*), 19

T

`target` (*class in migraphx*), 22
`tolist()` (*in module migraphx*), 21
`transposed()` (*in module migraphx*), 20
`type()` (*in module migraphx*), 19
`type_size()` (*in module migraphx*), 20