Python

Advanced Micro Devices <a href="https://">Disclaimer and Licens

May 18, 2024

CONTENTS

CHAPTER

ONE

INTRODUCTION

CHAPTER

TWO

GENERAL DESCRIPTION

hipBLASLt is a library that provides general matrix-matrix operations with a flexible API and extends funtionalities beyond traditional BLAS library. hipBLASLt is exposed APIs in HIP programming language with an underlying optimized generator as a backend kernel provider.

This library adds flexibility in matrix data layouts, input types, compute types, and also in choosing the algorithmic implementations and heuristics through parameter programmability. After a set of options for the intended GEMM operation are identified by the user, these options can be used repeatedly for different inputs.

The GEMM operation of hipBLASLt is performed by *hipblasLtMatmul()*. The equation is listed here:

$$D = Activation(alpha \cdot op(A) \cdot op(B) + beta \cdot op(C) + bias)$$

where $op(A)/op(B)$ refers to in-place operations such as transpose/non-transpose, and alpha, beta are scalars. Aciti-vation function supports Gelu, Relu. Bias vector match matrix D rows and broadcast to all D columns.

The code is open and hosted here: https://github.com/ROCmSoftwarePlatform/hipBLASLt

CHAPTER

THREE

HIPBLASLT LOGGING

The hipBLASLt logging mechanism can be enabled by setting the following environment variables before launching the target application:

HIPBLASLT_LOG_LEVEL=<level> - while level is one of the following levels:

“0” - Off - logging is disabled (default)
“1” - Error - only errors will be logged
“2” - Trace - API calls that launch HIP kernels will log their parameters and important information
“3” - Hints - hints that can potentially improve the application’s performance
“4” - Info - provides general information about the library execution, may contain details about heuristic status
“5” - API Trace - API calls will log their parameter and important information

HIPBLASLT_LOG_MASK=<mask> - while mask is a combination of the following masks:

“0” - Off
“1” - Error
“2” - Trace
“4” - Hints
“8” - Info
“16” - API Trace
“32” - Bench

HIPBLASLT_LOG_FILE=<file_name> - while file name is a path to a logging file. File name may contain %i, that will be replaced with the process ID. For example, “<file_name>_%i.log”. If HIPBLASLT_LOG_FILE is not defined, the log messages are printed to stdout.

CHAPTER

FOUR

HEURISTICS CACHE

hipBLASLt uses heuristics to pick the most suitable matmul kernel for execution based on the problem sizes, GPU configuration, and other parameters. This requires performing some computations on the host CPU, which could take tens of microseconds. To overcome this overhead, it is recommended to query the heuristics once using *hipblasLtMatmulAlgoGetHeuristic()* and then reuse the result for subsequent computations using *hipblasLtMatmul()*.

CHAPTER

FIVE

HIPBLASLT EXTENSIONS

See extension reference page for more information.

5.1 Building and Installing

5.1.1 Prerequisites

- A ROCm enabled platform, more information ROCm Documentation.
- A compatible version of hipBLAS

5.1.2 Installing pre-built packages

Download pre-built packages either from ROCm's package servers or by clicking the GitHub releases tab and manually downloading, which could be newer. Release notes are available for each release on the releases tab.

5.1.3 hipBLASLt build

Build library dependencies + library

The root of this repository has a helper bash script *install.sh* to build and install hipBLASLt with a single command. It does take a lot of options and hard-codes configuration that can be specified through invoking cmake directly, but it's a great way to get started quickly and can serve as an example of how to build/install. A few commands in the script need sudo access so that it may prompt you for a password.

Typical uses of install.sh to build (library dependencies + library) are in the table below.

Command	Description
`./install.sh -h`	Help information.
`./install.sh -d`	Build library dependencies and library in your local directory. The -d flag only needs to be used once. For subsequent invocations of install.sh it is not necessary to rebuild the dependencies.
`./install.sh`	Build library in your local directory. It is assumed dependencies have been built.
`./install.sh -i`	Build library, then build and install hipBLASLt package in */opt/rocm/hipblaslt*. You will be prompted for sudo access. This will install for all users. If you want to keep hipBLASLt in your local directory, you do not need the -i flag.

Build library dependencies + client dependencies + library + client

The client contains executables in the table below.

executable name	description
hipblaslt-test	runs Google Tests to test the library
hipblaslt-bench	executable to benchmark or test individual functions

Common uses of install.sh to build (dependencies + library + client) are in the table below.

Command	Description
`./install.sh -h`	Help information.
`./install.sh -dc`	Build library dependencies, client dependencies, library, and client in your local directory. The -d flag only needs to be used once. For subsequent invocations of install.sh it is not necessary to rebuild the dependencies.
`./install.sh -c`	Build library and client in your local directory. It is assumed the dependencies have been built.
`./install.sh -idc`	Build library dependencies, client dependencies, library, client, then build and install the hipBLASLt package. You will be prompted for sudo access. It is expected that if you want to install for all users you use the -i flag. If you want to keep hipBLASLt in your local directory, you do not need the -i flag.
`./install.sh -ic`	Build and install hipBLASLt package, and build the client. You will be prompted for sudo access. This will install for all users. If you want to keep hipBLASLt in your local directory, you do not need the -i flag.

5.1.4 Dependencies

Dependencies are listed in the script install.sh. Use `install.sh` with `-d` option to install dependencies. CMake has a minimum version requirement listed in the file install.sh. See –cmake_install flag in install.sh to upgrade automatically.

5.1.5 Manual build (all supported platforms)

This section has useful information on how to configure cmake and manually build.

Dependencies For Building Library

Build Library Using Individual Commands

Build Library + Tests + Benchmarks + Samples Using Individual Commands

The repository contains source for clients that serve as samples, tests and benchmarks. Clients source can be found in the clients subdir.

Dependencies (only necessary for hipBLASLt clients)

The hipBLASLt samples have no external dependencies, but our unit test and benchmarking applications do. These clients introduce the following dependencies:

- lapack, lapack itself brings a dependency on a fortran compiler
- googletest

Unfortunately, googletest and lapack are not as easy to install. Many distros do not provide a googletest package with pre-compiled libraries, and the lapack packages do not have the necessary cmake config files for cmake to configure linking the cblas library. hipBLASLt provide a cmake script that builds the above dependencies from source. This is an optional step; users can provide their own builds of these dependencies and help cmake find them by setting the CMAKE_PREFIX_PATH definition. The following is a sequence of steps to build dependencies and install them to the cmake default /usr/local.

(optional, one time only)

Once dependencies are available on the system, it is possible to configure the clients to build. This requires a few extra cmake flags to the library cmake configure script. If the dependencies are not installed into system defaults (like /usr/local), you should pass the CMAKE_PREFIX_PATH to cmake to help find them.

5.2 Library Source Code Organization

The hipBLASLt code is split into two major parts:

- The *library* directory contains all source code for the library.
- The *clients* directory contains all test code and code to build clients.
- Infrastructure

5.2.1 The *library* directory

library/include

Contains C98 include files for the external API. These files also contain Doxygen comments that document the API.

library/src/amd_detail

Implementation of hipBLASLt interface compatible with rocBLASLt APIs.

library/src/include

Internal include files for:

- Converting C++ exceptions to hipBLAS status.

5.2.2 The *clients* directory

clients/samples

Sample code for calling hipBLASLt functions.

5.2.3 Infrastructure

- CMake is used to build and package hipBLASLt. There are CMakeLists.txt files throughout the code.
- Doxygen/Breathe/Sphinx/ReadTheDocs are used to produce documentation. Content for the documentation is from:
  - Doxygen comments in include files in the directory library/include
  - files in the directory docs/source.
- Jenkins is used to automate Continuous Integration testing.
- clang-format is used to format C++ code.

5.3 hipBLASLt Datatypes Reference

5.3.1 hipblasLtEpilogue_t

enum **hipblasLtEpilogue_t**

Specify the enum type to set the postprocessing options for the epilogue.

Values:

enumerator **HIPBLASLT_EPILOGUE_DEFAULT**

No special postprocessing, just scale and quantize the results if necessary.

enumerator **HIPBLASLT_EPILOGUE_RELU**

Apply ReLU point-wise transform to the results:(x:=max(x, 0))

enumerator **HIPBLASLT_EPILOGUE_BIAS**

Apply (broadcast) bias from the bias vector. Bias vector length must match matrix D rows, and it must be packed (such as stride between vector elements is 1). Bias vector is broadcast to all columns and added before applying the final postprocessing.

enumerator **HIPBLASLT_EPILOGUE_RELU_BIAS**

Apply bias and then ReLU transform.

enumerator **HIPBLASLT_EPILOGUE_GELU**

Apply GELU point-wise transform to the results (x:=GELU(x)).

enumerator **HIPBLASLT_EPILOGUE_GELU_BIAS**

Apply Bias and then GELU transform.

enumerator **HIPBLASLT_EPILOGUE_GELU_AUX**

Output GEMM results before applying GELU transform.

enumerator **HIPBLASLT_EPILOGUE_GELU_AUX_BIAS**

Output GEMM results after applying bias but before applying GELU transform.

enumerator **HIPBLASLT_EPILOGUE_DGELU**

Apply gradient GELU transform. Requires additional aux input.

enumerator **HIPBLASLT_EPILOGUE_DGELU_BGRAD**

Apply gradient GELU transform and bias gradient to the results. Requires additional aux input.

enumerator **HIPBLASLT_EPILOGUE_BGRADA**

Apply bias gradient to A and output gemm result.

enumerator **HIPBLASLT_EPILOGUE_BGRADB**

Apply bias gradient to B and output gemm result.

5.3.2 hipblasLtPointerMode_t

enum **hipblasLtPointerMode_t**

Pointer mode to use for alpha.

Values:

enumerator **HIPBLASLT_POINTER_MODE_HOST**

enumerator **HIPBLASLT_POINTER_MODE_DEVICE**

targets host memory

enumerator **HIPBLASLT_POINTER_MODE_ALPHA_DEVICE_VECTOR_BETA_HOST**

targets device memory

5.3.3 hipblasLtHandle_t

typedef void *hipblasLtHandle_t

Handle to the hipBLASLt library context queue.

The hipblasLtHandle_t type is a pointer type to an opaque structure holding the hipBLASLt library context. Use the following functions to manipulate this library context:

hipblasLtCreate(): To initialize the hipBLASLt library context and return a handle to an opaque structure holding the hipBLASLt library context. *hipblasLtDestroy()*: To destroy a previously created hipBLASLt library context descriptor and release the resources.

5.3.4 hipblasLtMatmulAlgo_t

struct hipblasLtMatmulAlgo_t

Description of the matrix multiplication algorithm.

This is an opaque structure holding the description of the matrix multiplication algorithm. This structure can be trivially serialized and later restored for use with the same version of hipBLASLt library to save on selecting the right configuration again.

5.3.5 hipblasLtMatmulDesc_t

typedef hipblasLtMatmulDescOpaque_t *hipblasLtMatmulDesc_t

Descriptor of the matrix multiplication operation.

This is a pointer to an opaque structure holding the description of the matrix multiplication operation *hipblasLtMatmul()*. Use the following functions to manipulate this descriptor: *hipblasLtMatmulDescCreate()*: To create one instance of the descriptor. *hipblasLtMatmulDescDestroy()*: To destroy a previously created descriptor and release the resources.

5.3.6 hipblasLtMatmulDescAttributes_t

enum hipblasLtMatmulDescAttributes_t

Specify the attributes that define the specifics of the matrix multiply operation.

Values:

enumerator HIPBLASLT_MATMUL_DESC_TRANSA

Specifies the type of transformation operation that should be performed on matrix A. Default value is HIPBLAS_OP_N (for example, non-transpose operation). See hipblasOperation_t. Data Type:int32_t

enumerator HIPBLASLT_MATMUL_DESC_TRANSB

Specifies the type of transformation operation that should be performed on matrix B. Default value is HIPBLAS_OP_N (for example, non-transpose operation). See hipblasOperation_t. Data Type:int32_t

enumerator **HIPBLASLT_MATMUL_DESC_EPILOGUE**

Epilogue function. See hipblasLtEpilogue_t. Default value is: HIPBLASLT_EPILOGUE_DEFAULT. Data Type: uint32_t

enumerator **HIPBLASLT_MATMUL_DESC_BIAS_POINTER**

Bias or Bias gradient vector pointer in the device memory. Data Type:void* /const void*

enumerator **HIPBLASLT_MATMUL_DESC_BIAS_DATA_TYPE**

Type of the bias vector in the device memory. Can be set same as D matrix type or Scale type. Bias case: see HIPBLASLT_EPILOGUE_BIAS. Data Type:int32_t based on hipDataType

enumerator **HIPBLASLT_MATMUL_DESC_A_SCALE_POINTER**

Device pointer to the scale factor value that converts data in matrix A to the compute data type range. The scaling factor must have the same type as the compute type. If not specified, or set to NULL, the scaling factor is assumed to be 1. If set for an unsupported matrix data, scale, and compute type combination, calling *hipblasLtMatmul()* will return HIPBLAS_INVALID_VALUE. Default value: NULL Data Type: void* /const void*

enumerator **HIPBLASLT_MATMUL_DESC_B_SCALE_POINTER**

Equivalent to HIPBLASLT_MATMUL_DESC_A_SCALE_POINTER for matrix B. Default value: NULL Type: void* /const void*

enumerator **HIPBLASLT_MATMUL_DESC_C_SCALE_POINTER**

Equivalent to HIPBLASLT_MATMUL_DESC_A_SCALE_POINTER for matrix C. Default value: NULL Type: void* /const void*

enumerator **HIPBLASLT_MATMUL_DESC_D_SCALE_POINTER**

Equivalent to HIPBLASLT_MATMUL_DESC_A_SCALE_POINTER for matrix D. Default value: NULL Type: void* /const void*

enumerator **HIPBLASLT_MATMUL_DESC_EPILOGUE_AUX_SCALE_POINTER**

Equivalent to HIPBLASLT_MATMUL_DESC_A_SCALE_POINTER for matrix AUX. Default value: NULL Type: void* /const void*

enumerator **HIPBLASLT_MATMUL_DESC_EPILOGUE_AUX_POINTER**

Epilogue auxiliary buffer pointer in the device memory. Data Type:void* /const void*

enumerator **HIPBLASLT_MATMUL_DESC_EPILOGUE_AUX_LD**

The leading dimension of the epilogue auxiliary buffer pointer in the device memory. Data Type:int64_t

enumerator **HIPBLASLT_MATMUL_DESC_EPILOGUE_AUX_BATCH_STRIDE**

The batch stride of the epilogue auxiliary buffer pointer in the device memory. Data Type:int64_t

enumerator **HIPBLASLT_MATMUL_DESC_POINTER_MODE**

Specifies alpha and beta are passed by reference, whether they are scalars on the host or on the device, or device vectors. Default value is: HIPBLASLT_POINTER_MODE_HOST (i.e., on the host). Data Type: int32_t based on hipblasLtPointerMode_t

enumerator **HIPBLASLT_MATMUL_DESC_MAX**

5.3.7 hipblasLtMatmulHeuristicResult_t

struct **hipblasLtMatmulHeuristicResult_t**

Description of the matrix multiplication algorithm.

This is a descriptor that holds the configured matrix multiplication algorithm descriptor and its runtime properties. This structure can be trivially serialized and later restored for use with the same version of hipBLASLt library to save on selecting the right configuration again.

5.3.8 hipblasLtMatmulPreference_t

typedef hipblasLtMatmulPreferenceOpaque_t ***hipblasLtMatmulPreference_t**

Descriptor of the matrix multiplication preference.

This is a pointer to an opaque structure holding the description of the preferences for *hipblasLtMatmulAlgoGetHeuristic()* configuration. Use the following functions to manipulate this descriptor: *hipblasLtMatmulPreferenceCreate()*: To create one instance of the descriptor. *hipblasLtMatmulPreferenceDestroy()*: To destroy a previously created descriptor and release the resources.

5.3.9 hipblasLtMatmulPreferenceAttributes_t

enum **hipblasLtMatmulPreferenceAttributes_t**

It is an enumerated type used to apply algorithm search preferences while fine-tuning the heuristic function.

Values:

enumerator **HIPBLASLT_MATMUL_PREF_SEARCH_MODE**

Search mode. Data Type: uint32_t

enumerator **HIPBLASLT_MATMUL_PREF_MAX_WORKSPACE_BYTES**

Maximum allowed workspace memory. Default is 0 (no workspace memory allowed). Data Type: uint64_t

enumerator **HIPBLASLT_MATMUL_PREF_MAX**

5.3.10 hipblasLtMatrixLayout_t

typedef hipblasLtMatrixLayoutOpaque_t ***hipblasLtMatrixLayout_t**

Descriptor of the matrix layout.

This is a pointer to an opaque structure holding the description of a matrix layout. Use the following functions to manipulate this descriptor: *hipblasLtMatrixLayoutCreate()*: To create one instance of the descriptor. *hipblasLtMatrixLayoutDestroy()*: To destroy a previously created descriptor and release the resources.

5.3.11 hipblasLtMatrixLayoutAttribute_t

enum **hipblasLtMatrixLayoutAttribute_t**

Specify the attributes that define the details of the matrix.

Values:

enumerator **HIPBLASLT_MATRIX_LAYOUT_BATCH_COUNT**

Number of batch of this matrix. Default value is 1. Data Type: int32_t

enumerator **HIPBLASLT_MATRIX_LAYOUT_STRIDED_BATCH_OFFSET**

Stride (in elements) to the next matrix for the strided batch operation. Default value is 0. Data Type: int64_t

enumerator **HIPBLASLT_MATRIX_LAYOUT_TYPE**

Data type, see hipDataType.

uint32_t

enumerator **HIPBLASLT_MATRIX_LAYOUT_ORDER**

Memory order of the data, see hipblasLtOrder_t.

int32_t, default: HIPBLASLT_ORDER_COL

enumerator **HIPBLASLT_MATRIX_LAYOUT_ROWS**

Number of rows.

Usually only values that can be expressed as int32_t are supported.

uint64_t

enumerator **HIPBLASLT_MATRIX_LAYOUT_COLS**

Number of columns.

Usually only values that can be expressed as int32_t are supported.

uint64_t

enumerator **HIPBLASLT_MATRIX_LAYOUT_LD**

Matrix leading dimension.

For HIPBLASLT_ORDER_COL this is stride (in elements) of matrix column, for more details and documentation for other memory orders see documentation for hipblasLtOrder_t values.

Currently only non-negative values are supported, must be large enough so that matrix memory locations are not overlapping (e.g. greater or equal to HIPBLASLT_MATRIX_LAYOUT_ROWS in case of HIPBLASLT_ORDER_COL).

int64_t;

5.3.12 hipblasLtMatrixTransformDesc_t

typedef hipblasLtMatrixTransformDescOpaque_t *hipblasLtMatrixTransformDesc_t

Opaque descriptor for *hipblasLtMatrixTransform()* operation details.

The hipblasLtMatrixTransformDesc_t is a pointer to an opaque structure holding the description of a matrix transformation operation. *hipblasLtMatrixTransformDescCreate()*: To create one instance of the descriptor. *hipblasLtMatrixTransformDescDestroy()*: To destroy a previously created descriptor and release the resources.

5.4 hipBLASLt API Reference

5.4.1 hipblasLtCreate()

hipblasStatus_t **hipblasLtCreate**(*hipblasLtHandle_t* *handle)

Create a hipblaslt handle.

This function initializes the hipBLASLt library and creates a handle to an opaque structure holding the hipBLASLt library context. It allocates light hardware resources on the host and device, and must be called prior to making any other hipBLASLt library calls. The hipBLASLt library context is tied to the current CUDA device. To use the library on multiple devices, one hipBLASLt handle should be created for each device.

Parameters
handle – **[out]** Pointer to the allocated hipBLASLt handle for the created hipBLASLt context.

Return values

- **HIPBLAS_STATUS_SUCCESS** – The allocation completed successfully.
- **HIPBLAS_STATUS_INVALID_VALUE** – handle == NULL.

5.4.2 hipblasLtDestroy()

hipblasStatus_t **hipblasLtDestroy**(const *hipblasLtHandle_t* handle)

Destory a hipblaslt handle.

This function releases hardware resources used by the hipBLASLt library. This function is usually the last call with a particular handle to the hipBLASLt library. Because *hipblasLtCreate()* allocates some internal resources and the release of those resources by calling *hipblasLtDestroy()* will implicitly call hipDeviceSynchronize(), it is recommended to minimize the number of *hipblasLtCreate()*/hipblasLtDestroy() occurrences.

Parameters
handle – **[in]** Pointer to the hipBLASLt handle to be destroyed.

Return values

- **HIPBLAS_STATUS_SUCCESS** – The hipBLASLt context was successfully destroyed.
- **HIPBLAS_STATUS_NOT_INITIALIZED** – The hipBLASLt library was not initialized.
- **HIPBLAS_STATUS_INVALID_VALUE** – handle == NULL.

5.4.3 hipblasLtMatrixLayoutCreate()

hipblasStatus_t **hipblasLtMatrixLayoutCreate**(*hipblasLtMatrixLayout_t* *matLayout, hipDataType type, uint64_t rows, uint64_t cols, int64_t ld)

Create a matrix layout descriptor.

This function creates a matrix layout descriptor by allocating the memory needed to hold its opaque structure.

Parameters

- **matLayout** – **[out]** Pointer to the structure holding the matrix layout descriptor created by this function. see *hipblasLtMatrixLayout_t* .
- **type** – **[in]** Enumerant that specifies the data precision for the matrix layout descriptor this function creates. See hipDataType.
- **rows** – **[in]** Number of rows of the matrix.
- **cols** – **[in]** Number of columns of the matrix.
- **ld** – **[in]** The leading dimension of the matrix. In column major layout, this is the number of elements to jump to reach the next column. Thus ld >= m (number of rows).

Return values

- **HIPBLAS_STATUS_SUCCESS** – If the descriptor was created successfully.
- **HIPBLAS_STATUS_ALLOC_FAILED** – If the memory could not be allocated.

5.4.4 hipblasLtMatrixLayoutDestroy()

hipblasStatus_t **hipblasLtMatrixLayoutDestroy**(const *hipblasLtMatrixLayout_t* matLayout)

Destory a matrix layout descriptor.

This function destroys a previously created matrix layout descriptor object.

Parameters

matLayout – **[in]** Pointer to the structure holding the matrix layout descriptor that should be destroyed by this function. see *hipblasLtMatrixLayout_t* .

Return values

HIPBLAS_STATUS_SUCCESS – If the operation was successful.

5.4.5 hipblasLtMatrixLayoutSetAttribute()

hipblasStatus_t **hipblasLtMatrixLayoutSetAttribute**(*hipblasLtMatrixLayout_t* matLayout, *hipblasLtMatrixLayoutAttribute_t* attr, const void *buf, size_t sizeInBytes)

Set attribute to a matrix descriptor.

This function sets the value of the specified attribute belonging to a previously created matrix descriptor.

Parameters

- **matLayout** – **[in]** Pointer to the previously created structure holding the matrix mdescriptor queried by this function. See *hipblasLtMatrixLayout_t*.
- **attr** – **[in]** The attribute that will be set by this function. See *hipblasLtMatrixLayoutAttribute_t*.

- **buf** – **[in]** The value to which the specified attribute should be set.
- **sizeInBytes** – **[in]** Size of buf buffer (in bytes) for verification.

Return values

- **HIPBLAS_STATUS_SUCCESS** – If the attribute was set successfully..
- **HIPBLAS_STATUS_INVALID_VALUE** – If `buf` is NULL or `sizeInBytes` doesn't match the size of the internal storage for the selected attribute.

5.4.6 hipblasLtMatrixLayoutGetAttribute()

hipblasStatus_t **hipblasLtMatrixLayoutGetAttribute**(*hipblasLtMatrixLayout_t* matLayout, *hipblasLtMatrixLayoutAttribute_t* attr, void *buf, size_t sizeInBytes, size_t *sizeWritten)

Query attribute from a matrix descriptor.

This function returns the value of the queried attribute belonging to a previously created matrix descriptor.

Parameters

- **matLayout** – **[in]** Pointer to the previously created structure holding the matrix descriptor queried by this function. See *hipblasLtMatrixLayout_t*.
- **attr** – **[in]** The attribute that will be retrieved by this function. See *hipblasLtMatrixLayoutAttribute_t*.
- **buf** – **[out]** Memory address containing the attribute value retrieved by this function.
- **sizeInBytes** – **[in]** Size of `buf` buffer (in bytes) for verification.
- **sizeWritten** – **[out]** Valid only when the return value is HIPBLAS_STATUS_SUCCESS. If sizeInBytes is non-zero: then sizeWritten is the number of bytes actually written; if sizeInBytes is 0: then sizeWritten is the number of bytes needed to write full contents.

Return values

- **HIPBLAS_STATUS_SUCCESS** – If attribute's value was successfully written to user memory.
- **HIPBLAS_STATUS_INVALID_VALUE** – If `sizeInBytes` is 0 and `sizeWritten` is NULL, or if `sizeInBytes` is non-zero and `buf` is NULL, or `sizeInBytes` doesn't match size of internal storage for the selected attribute.

5.4.7 hipblasLtMatmulDescCreate()

hipblasStatus_t **hipblasLtMatmulDescCreate**(*hipblasLtMatmulDesc_t* *matmulDesc, hipblasComputeType_t computeType, hipDataType scaleType)

Create a matrix multiply descriptor.

This function creates a matrix multiply descriptor by allocating the memory needed to hold its opaque structure.

Parameters

- **matmulDesc** – **[out]** Pointer to the structure holding the matrix multiply descriptor created by this function. See *hipblasLtMatmulDesc_t* .
- **computeType** – **[in]** Enumerant that specifies the data precision for the matrix multiply descriptor this function creates. See hipblasComputeType_t .

- **scaleType** – **[in]** Enumerant that specifies the data precision for the matrix transform descriptor this function creates. See hipDataType.

Return values

- **HIPBLAS_STATUS_SUCCESS** – If the descriptor was created successfully.
- **HIPBLAS_STATUS_ALLOC_FAILED** – If the memory could not be allocated.

5.4.8 hipblasLtMatmulDescDestroy()

hipblasStatus_t **hipblasLtMatmulDescDestroy**(const *hipblasLtMatmulDesc_t* matmulDesc)

Destory a matrix multiply descriptor.

This function destroys a previously created matrix multiply descriptor object.

Parameters

matmulDesc – **[in]** Pointer to the structure holding the matrix multiply descriptor that should be destroyed by this function. See *hipblasLtMatmulDesc_t* .

Return values

HIPBLAS_STATUS_SUCCESS – If operation was successful.

5.4.9 hipblasLtMatmulDescSetAttribute()

hipblasStatus_t **hipblasLtMatmulDescSetAttribute**(*hipblasLtMatmulDesc_t* matmulDesc, *hipblasLtMatmulDescAttributes_t* attr, const void *buf, size_t sizeInBytes)

Set attribute to a matrix multiply descriptor.

This function sets the value of the specified attribute belonging to a previously created matrix multiply descriptor.

Parameters

- **matmulDesc** – **[in]** Pointer to the previously created structure holding the matrix multiply descriptor queried by this function. See *hipblasLtMatmulDesc_t*.
- **attr** – **[in]** The attribute that will be set by this function. See *hipblasLtMatmulDescAttributes_t*.
- **buf** – **[in]** The value to which the specified attribute should be set.
- **sizeInBytes** – **[in]** Size of buf buffer (in bytes) for verification.

Return values

- **HIPBLAS_STATUS_SUCCESS** – If the attribute was set successfully..
- **HIPBLAS_STATUS_INVALID_VALUE** – If `buf` is NULL or `sizeInBytes` doesn't match the size of the internal storage for the selected attribute.

5.4.10 hipblasLtMatmulDescGetAttribute()

hipblasStatus_t **hipblasLtMatmulDescGetAttribute**(*hipblasLtMatmulDesc_t* matmulDesc, *hipblasLtMatmulDescAttributes_t* attr, void *buf, size_t sizeInBytes, size_t *sizeWritten)

Query attribute from a matrix multiply descriptor.

This function returns the value of the queried attribute belonging to a previously created matrix multiply descriptor.

Parameters

- **matmulDesc** – **[in]** Pointer to the previously created structure holding the matrix multiply descriptor queried by this function. See *hipblasLtMatmulDesc_t*.
- **attr** – **[in]** The attribute that will be retrieved by this function. See *hipblasLtMatmulDescAttributes_t*.
- **buf** – **[out]** Memory address containing the attribute value retrieved by this function.
- **sizeInBytes** – **[in]** Size of `buf` buffer (in bytes) for verification.
- **sizeWritten** – **[out]** Valid only when the return value is HIPBLAS_STATUS_SUCCESS. If sizeInBytes is non-zero: then sizeWritten is the number of bytes actually written; if sizeInBytes is 0: then sizeWritten is the number of bytes needed to write full contents.

Return values

- **HIPBLAS_STATUS_SUCCESS** – If attribute's value was successfully written to user memory.
- **HIPBLAS_STATUS_INVALID_VALUE** – If `sizeInBytes` is 0 and `sizeWritten` is NULL, or if `sizeInBytes` is non-zero and `buf` is NULL, or `sizeInBytes` doesn't match size of internal storage for the selected attribute.

5.4.11 hipblasLtMatmulPreferenceCreate()

hipblasStatus_t **hipblasLtMatmulPreferenceCreate**(*hipblasLtMatmulPreference_t* *pref)

Create a preference descriptor.

This function creates a matrix multiply heuristic search preferences descriptor by allocating the memory needed to hold its opaque structure.

Parameters

pref – **[out]** Pointer to the structure holding the matrix multiply preferences descriptor created by this function. see *hipblasLtMatmulPreference_t* .

Return values

- **HIPBLAS_STATUS_SUCCESS** – If the descriptor was created successfully.
- **HIPBLAS_STATUS_ALLOC_FAILED** – If memory could not be allocated.

5.4.12 hipblasLtMatmulPreferenceDestroy()

hipblasStatus_t **hipblasLtMatmulPreferenceDestroy**(const *hipblasLtMatmulPreference_t* pref)

Destory a preferences descriptor.

This function destroys a previously created matrix multiply preferences descriptor object.

Parameters

pref – **[in]** Pointer to the structure holding the matrix multiply preferences descriptor that should be destroyed by this function. See *hipblasLtMatmulPreference_t* .

Return values

HIPBLAS_STATUS_SUCCESS – If operation was successful.

5.4.13 hipblasLtMatmulPreferenceSetAttribute()

hipblasStatus_t **hipblasLtMatmulPreferenceSetAttribute**(*hipblasLtMatmulPreference_t* pref, *hipblasLtMatmulPreferenceAttributes_t* attr, const void *buf, size_t sizeInBytes)

Set attribute to a preference descriptor.

This function sets the value of the specified attribute belonging to a previously created matrix multiply preferences descriptor.

Parameters

- **pref** – **[in]** Pointer to the previously created structure holding the matrix multiply preferences descriptor queried by this function. See *hipblasLtMatmulPreference_t*
- **attr** – **[in]** The attribute that will be set by this function. See *hipblasLtMatmulPreferenceAttributes_t*.
- **buf** – **[in]** The value to which the specified attribute should be set.
- **sizeInBytes** – **[in]** Size of `buf` buffer (in bytes) for verification.

Return values

- **HIPBLAS_STATUS_SUCCESS** – If the attribute was set successfully..
- **HIPBLAS_STATUS_INVALID_VALUE** – If `buf` is NULL or `sizeInBytes` doesn't match the size of the internal storage for the selected attribute.

5.4.14 hipblasLtMatmulPreferenceGetAttribute()

hipblasStatus_t **hipblasLtMatmulPreferenceGetAttribute**(*hipblasLtMatmulPreference_t* pref, *hipblasLtMatmulPreferenceAttributes_t* attr, void *buf, size_t sizeInBytes, size_t *sizeWritten)

Query attribute from a preference descriptor.

This function returns the value of the queried attribute belonging to a previously created matrix multiply heuristic search preferences descriptor.

Parameters

- **pref** – **[in]** Pointer to the previously created structure holding the matrix multiply heuristic search preferences descriptor queried by this function. See *hipblasLtMatmulPreference_t*.

- **attr** – **[in]** The attribute that will be retrieved by this function. See *hipblasLtMatmulPreferenceAttributes_t*.
- **buf** – **[out]** Memory address containing the attribute value retrieved by this function.
- **sizeInBytes** – **[in]** Size of `buf` buffer (in bytes) for verification.
- **sizeWritten** – **[out]** Valid only when the return value is HIPBLAS_STATUS_SUCCESS. If sizeInBytes is non-zero: then sizeWritten is the number of bytes actually written; if sizeInBytes is 0: then sizeWritten is the number of bytes needed to write full contents.

Return values

- **HIPBLAS_STATUS_SUCCESS** – If attribute's value was successfully written to user memory.
- **HIPBLAS_STATUS_INVALID_VALUE** – If `sizeInBytes` is 0 and `sizeWritten` is NULL, or if `sizeInBytes` is non-zero and `buf` is NULL, or `sizeInBytes` doesn't match size of internal storage for the selected attribute.

5.4.15 hipblasLtMatmulAlgoGetHeuristic()

hipblasStatus_t **hipblasLtMatmulAlgoGetHeuristic**(*hipblasLtHandle_t* handle, *hipblasLtMatmulDesc_t* matmulDesc, *hipblasLtMatrixLayout_t* Adesc, *hipblasLtMatrixLayout_t* Bdesc, *hipblasLtMatrixLayout_t* Cdesc, *hipblasLtMatrixLayout_t* Ddesc, *hipblasLtMatmulPreference_t* pref, int requestedAlgoCount, *hipblasLtMatmulHeuristicResult_t* heuristicResultsArray[], int *returnAlgoCount)

Retrieve the possible algorithms.

This function retrieves the possible algorithms for the matrix multiply operation *hipblasLtMatmul()* function with the given input matrices A, B and C, and the output matrix D. The output is placed in heuristicResultsArray[] in the order of increasing estimated compute time.

Parameters

- **handle** – **[in]** Pointer to the allocated hipBLASLt handle for the hipBLASLt context. See *hipblasLtHandle_t* .
- **matmulDesc** – **[in]** Handle to a previously created matrix multiplication descriptor of type *hipblasLtMatmulDesc_t* .
- **Adesc, Bdesc, Cdesc, Ddesc** – **[in]** Handles to the previously created matrix layout descriptors of the type *hipblasLtMatrixLayout_t* .
- **pref** – **[in]** Pointer to the structure holding the heuristic search preferences descriptor. See *hipblasLtMatmulPreference_t* .
- **requestedAlgoCount** – **[in]** Size of the `heuristicResultsArray` (in elements). This is the requested maximum number of algorithms to return.
- **heuristicResultsArray[]** – **[out]** Array containing the algorithm heuristics and associated runtime characteristics, returned by this function, in the order of increasing estimated compute time.
- **returnAlgoCount** – **[out]** Number of algorithms returned by this function. This is the number of `heuristicResultsArray` elements written.

Return values

- **HIPBLAS_STATUS_SUCCESS** – If query was successful. Inspect heuristicResultsArray[0 to (returnAlgoCount -1)].state for the status of the results.
- **HIPBLAS_STATUS_NOT_SUPPORTED** – If no heuristic function available for current configuration.
- **HIPBLAS_STATUS_INVALID_VALUE** – If `requestedAlgoCount` is less or equal to zero.

5.4.16 hipblasLtMatmul()

hipblasStatus_t **hipblasLtMatmul**(*hipblasLtHandle_t* handle, *hipblasLtMatmulDesc_t* matmulDesc, const void *alpha, const void *A, *hipblasLtMatrixLayout_t* Adesc, const void *B, *hipblasLtMatrixLayout_t* Bdesc, const void *beta, const void *C, *hipblasLtMatrixLayout_t* Cdesc, void *D, *hipblasLtMatrixLayout_t* Ddesc, const *hipblasLtMatmulAlgo_t* *algo, void *workspace, size_t workspaceSizeInBytes, hipStream_t stream)

Retrieve the possible algorithms.

This function computes the matrix multiplication of matrices A and B to produce the output matrix D, according to the following operation: `D = alpha*( A *B) + beta*( C )`, where `A`, `B`, and `C` are input matrices, and `alpha` and `beta` are input scalars. Note: This function supports both in-place matrix multiplication (C == D and Cdesc == Ddesc) and out-of-place matrix multiplication (C != D, both matrices must have the same data type, number of rows, number of columns, batch size, and memory order). In the out-of-place case, the leading dimension of C can be different from the leading dimension of D. Specifically the leading dimension of C can be 0 to achieve row or column broadcast. If Cdesc is omitted, this function assumes it to be equal to Ddesc.

Parameters

- **handle** – **[in]** Pointer to the allocated hipBLASLt handle for the hipBLASLt context. See *hipblasLtHandle_t* .
- **matmulDesc** – **[in]** Handle to a previously created matrix multiplication descriptor of type *hipblasLtMatmulDesc_t* .
- **alpha, beta** – **[in]** Pointers to the scalars used in the multiplication.
- **Adesc, Bdesc, Cdesc, Ddesc** – **[in]** Handles to the previously created matrix layout descriptors of the type *hipblasLtMatrixLayout_t* .
- **A, B, C** – **[in]** Pointers to the GPU memory associated with the corresponding descriptors `Adesc`, `Bdesc` and `Cdesc` .
- **D** – **[out]** Pointer to the GPU memory associated with the descriptor `Ddesc` .
- **algo** – **[in]** Handle for matrix multiplication algorithm to be used. See *hipblasLtMatmulAlgo_t* . When NULL, an implicit heuristics query with default search preferences will be performed to determine actual algorithm to use.
- **workspace** – **[in]** Pointer to the workspace buffer allocated in the GPU memory. Pointer must be 16B aligned (that is, lowest 4 bits of address must be 0).
- **workspaceSizeInBytes** – **[in]** Size of the workspace.
- **stream** – **[in]** The HIP stream where all the GPU work will be submitted.

Return values

- **HIPBLAS_STATUS_SUCCESS** – If the operation completed successfully.
- **HIPBLAS_STATUS_EXECUTION_FAILED** – If HIP reported an execution error from the device.

- **HIPBLAS_STATUS_ARCH_MISMATCH** – If the configured operation cannot be run using the selected device.
- **HIPBLAS_STATUS_NOT_SUPPORTED** – If the current implementation on the selected device doesn't support the configured operation.
- **HIPBLAS_STATUS_INVALID_VALUE** – If the parameters are unexpectedly NULL, in conflict or in an impossible configuration. For example, when workspaceSizeInBytes is less than workspace required by the configured algo.
- **HIBLAS_STATUS_NOT_INITIALIZED** – If hipBLASLt handle has not been initialized.

Datatypes Supported:

hipblasLtMatmul supports the following computeType, scaleType, Atype/Btype, Ctype/Dtype and Bias Type:

computeType	scaleType/Bias Type	Atype/Btype	Ctype/Dtype
HIPBLAS_COMPUTE_32F	HIP_R_32F	HIP_R_32F	HIP_R_32F
HIPBLAS_COMPUTE_32F	HIP_R_32F	HIP_R_16F	HIP_R_16F
HIPBLAS_COMPUTE_32F	HIP_R_32F	HIP_R_16F	HIP_R_32F
HIPBLAS_COMPUTE_32F	HIP_R_32F	HIP_R_16BF	HIP_R_16BF

5.4.17 hipblasLtMatrixTransformDescCreate()

hipblasStatus_t **hipblasLtMatrixTransformDescCreate**(*hipblasLtMatrixTransformDesc_t* *transformDesc, hipDataType scaleType)

Create new matrix transform operation descriptor.

Return values

- **HIPBLAS_STATUS_ALLOC_FAILED** – if memory could not be allocated
- **HIPBLAS_STATUS_SUCCESS** – if desciptor was created successfully

5.4.18 hipblasLtMatrixTransformDescDestroy()

hipblasStatus_t **hipblasLtMatrixTransformDescDestroy**(*hipblasLtMatrixTransformDesc_t* transformDesc)

Destroy matrix transform operation descriptor.

Return values

HIPBLAS_STATUS_SUCCESS – if operation was successful

5.4.19 hipblasLtMatrixTransformDescSetAttribute()

hipblasStatus_t **hipblasLtMatrixTransformDescSetAttribute**(*hipblasLtMatrixTransformDesc_t* transformDesc, hipblasLtMatrixTransformDescAttributes_t attr, const void *buf, size_t sizeInBytes)

Set matrix transform operation descriptor attribute.

Parameters

- **transformDesc** – **[in]** The descriptor
- **attr** – **[in]** The attribute

- **buf** – **[in]** memory address containing the new value
- **sizeInBytes** – **[in]** size of buf buffer for verification (in bytes)

Return values

- **HIPBLAS_STATUS_INVALID_VALUE** – if buf is NULL or sizeInBytes doesn't match size of internal storage for selected attribute
- **HIPBLAS_STATUS_SUCCESS** – if attribute was set successfully

5.4.20 hipblasLtMatrixTransformDescGetAttribute()

hipblasStatus_t **hipblasLtMatrixTransformDescGetAttribute**(*hipblasLtMatrixTransformDesc_t* transformDesc, hipblasLtMatrixTransformDescAttributes_t attr, void *buf, size_t sizeInBytes, size_t *sizeWritten)

Matrix transform operation getter.

Get matrix transform operation descriptor attribute.

Parameters

- **transformDesc** – **[in]** The descriptor
- **attr** – **[in]** The attribute
- **buf** – **[out]** memory address containing the new value
- **sizeInBytes** – **[in]** size of buf buffer for verification (in bytes)
- **sizeWritten** – **[out]** only valid when return value is HIPBLAS_STATUS_SUCCESS. If sizeInBytes is non-zero: number of bytes actually written, if sizeInBytes is 0: number of bytes needed to write full contents

Return values

- **HIPBLAS_STATUS_INVALID_VALUE** – if sizeInBytes is 0 and sizeWritten is NULL, or if sizeInBytes is non-zero and buf is NULL or sizeInBytes doesn't match size of internal storage for selected attribute
- **HIPBLAS_STATUS_SUCCESS** – if attribute's value was successfully written to user memory

5.4.21 hipblasLtMatrixTransform()

hipblasStatus_t **hipblasLtMatrixTransform**(*hipblasLtHandle_t* lightHandle, *hipblasLtMatrixTransformDesc_t* transformDesc, const void *alpha, const void *A, *hipblasLtMatrixLayout_t* Adesc, const void *beta, const void *B, *hipblasLtMatrixLayout_t* Bdesc, void *C, *hipblasLtMatrixLayout_t* Cdesc, hipStream_t stream)

Matrix layout conversion helper.

Matrix layout conversion helper (C = alpha * op(A) + beta * op(B)), can be used to change memory order of data or to scale and shift the values.

Parameters

- **lightHandle** – **[in]** Pointer to the allocated hipBLASLt handle for the hipBLASLt context. See *hipblasLtHandle_t* .

- **transformDesc** – **[in]** Pointer to allocated matrix transform descriptor.
- **alpha** – **[in]** Pointer to scalar alpha, either pointer to host or device address.
- **A** – **[in]** Pointer to matrix A, must be pointer to device address.
- **Adesc** – **[in]** Pointer to layout for input matrix A.
- **beta** – **[in]** Pointer to scalar beta, either pointer to host or device address.
- **B** – **[in]** Pointer to layout for matrix B, must be pointer to device address
- **Bdesc** – **[in]** Pointer to layout for inputmatrix B.
- **C** – **[in]** Pointer to matrix C, must be pointer to device address
- **Cdesc** – **[out]** Pointer to layout for output matrix C.
- **stream** – **[in]** The HIP stream where all the GPU work will be submitted.

Return values

- **HIPBLAS_STATUS_NOT_INITIALIZED** – if hipBLASLt handle has not been initialized
- **HIPBLAS_STATUS_INVALID_VALUE** – if parameters are in conflict or in an impossible configuration; e.g. when A is not NULL, but Adesc is NULL
- **HIPBLAS_STATUS_NOT_SUPPORTED** – if current implementation on selected device doesn't support configured operation
- **HIPBLAS_STATUS_ARCH_MISMATCH** – if configured operation cannot be run using selected device
- **HIPBLAS_STATUS_EXECUTION_FAILED** – if HIP reported execution error from the device
- **HIPBLAS_STATUS_SUCCESS** – if the operation completed successfully

hipblasLtMatrixTransform supports the following Atype/Btype/Ctype and scaleType:

Atype/Btype/Ctype	scaleType
HIP_R_32F	HIP_R_32F
HIP_R_16F	HIP_R_32F/HIP_R_16F
HIP_R_16BF	HIP_R_32F
HIP_R_8I	HIP_R_32F
HIP_R_32I	HIP_R_32F

5.5 hipBLASLtExt Reference

5.5.1 hipBLASLtExt Datatypes Reference

GemmType

enum class `hipblaslt_ext::GemmType`

It is an enumerated type used to specific the type of the gemm problem in hipblasLtExt APIs.

Values:

enumerator **HIPBLASLT_GEMM**

enumerator **HIPBLASLT_GROUPED_GEMM**

GemmProblemType

struct **GemmProblemType**

hipblasLt extension ProblemType for gemm problems.

This structure sets the problem type of a gemm problem.

Public Members

hipblasOperation_t **op_a**

The A martix transpose.

hipblasOperation_t **op_b**

The B matrix transpose.

hipDataType **type_a**

The A matrix datatype.

hipDataType **type_b**

The B matrix datatype.

hipDataType **type_c**

The C matrix datatype.

hipDataType **type_d**

The D matrix datatype.

hipblasComputeType_t **type_compute**

The compute datatype.

GemmEpilogue

struct **GemmEpilogue**

hipblasLt extension Epilogue for gemm problems.

This structure sets the epilogue of a gemm problem.

Public Members

hipblasLtEpilogue_t **mode** = HIPBLASLT_EPILOGUE_DEFAULT

The mode of epilogue. Default is gemm.

hipDataType **bias_data_type** = HIPBLASLT_DATATYPE_INVALID

The bias datatype. Only works if mode is set to bias related epilogues.

int **aux_ld** = 0

The aux leading dimension. Only works if mode is set to aux related epilogues.

int **aux_stride** = 0

The aux batch stride. Only works if mode is set to aux related epilogues.

GemmInputs

struct **GemmInputs**

hipblasLt extension Inputs for gemm problems.

This structure sets the input pointers of a gemm problem.

Public Members

void ***a** = nullptr

The a matrix input pointer.

void ***b** = nullptr

The b matrix input pointer.

void ***c** = nullptr

The c matrix input pointer.

void ***d** = nullptr

The d matrix input pointer.

void ***alpha** = nullptr

The alpha value.

void ***beta** = nullptr

The beta value.

void ***bias** = nullptr

The bias input pointer.

void *scaleA = nullptr

The Scale A input pointer.

void *scaleB = nullptr

The Scale B input pointer.

void *scaleC = nullptr

The Scale C input pointer.

void *scaleD = nullptr

The Scale D input pointer.

void *scaleAux = nullptr

The Scale AUX input pointer.

void *scaleAlphaVec = nullptr

The scaleAlpha vector input pointer.

void *aux = nullptr

The aux input pointer.

5.5.2 hipBLASLtExt Class Reference

GemmPreference

class GemmPreference

hipblasLt extension preference for gemm problems.

Currently only supports setting max workspace size.

Public Functions

void setMaxWorkspaceBytes(size_t workspaceBytes)

This function sets the max workspace size.

Parameters

workspaceBytes – **[in]** Set the max workspace size in bytes.

const size_t getMaxWorkspaceBytes() const

This function returns the set max workspace size.

Return values

size_t – Returns the set max workspace size.

GemmInstance

class **GemmInstance**

hipblasLt extension instance for gemm problems.

Subclassed by *hipblaslt_ext::Gemm*, *hipblaslt_ext::GroupedGemm*

Public Functions

hipblasStatus_t **algoGetHeuristic**(const int requestedAlgoCount, const *GemmPreference* &pref, std::vector<*hipblasLtMatmulHeuristicResult_t*> &heuristicResults)

Retrieve the possible algorithms.

This function retrieves the possible algorithms for the matrix multiply operation *hipblasLtMatmul()* function with the given data and compute tpye. The output is placed in heuristicResult in the order of increasing estimated compute time.

Parameters

- **requestedAlgoCount** – **[in]** number of requested algorithms.
- **pref** – **[in]** hipblasLt extension preference for gemm problems.
- **heuristicResults** – **[out]** The algorithm heuristic vector.

Return values

- **HIPBLAS_STATUS_SUCCESS** – If query was successful. Inspect heuristicResults.size > 0, but may heuristicResults.size < requestedAlgoCount state for the status of the results.
- **HIPBLAS_STATUS_NOT_SUPPORTED** – If no heuristic function available for current configuration.
- **HIPBLAS_STATUS_INVALID_VALUE** – If no solution is found.

hipblasStatus_t **isAlgoSupported**(*hipblasLtMatmulAlgo_t* &algo, size_t &workspaceSizeInBytes)

Check if the algorithm supports the problem. (For hipblaslt extension API)

This function updates the problem saved inside the algorithm if the problem is supported. The required workspaceSizeInBytes is also returned.

Parameters

- **algo** – **[in]** The algorithm heuristic.
- **workspaceSizeInBytes** – **[out]** Return the required workspace size.

Return values

- **HIPBLAS_STATUS_SUCCESS** – If query was successful. The problem is supported by the algorithm. results.
- **HIPBLAS_STATUS_INVALID_VALUE** – The problem is not supported.

hipblasStatus_t **isAlgoSupported**(*hipblasLtMatmulAlgo_t* &algo, GemmTuning &tuning, size_t &workspaceSizeInBytes)

Check if the algorithm supports the problem. (For hipblaslt extension API)

This function updates the problem saved inside the algorithm if the problem is supported. The required workspaceSizeInBytes is also returned.

Parameters

- **algo** – **[in]** The algorithm heuristic.
- **tuning** – **[in]** The tuning parameters.
- **workspaceSizeInBytes** – **[out]** Return the required workspace size.

Return values

- **HIPBLAS_STATUS_SUCCESS** – If query was successful. The problem is supported by the algorithm. results.
- **HIPBLAS_STATUS_INVALID_VALUE** – The problem is not supported.

hipblasStatus_t **initialize**(const *hipblasLtMatmulAlgo_t* &algo, void *workspace, bool useUserArgs = true, hipStream_t stream = 0)

Create kernel arguments from a given *hipblaslt_ext::GemmInstance*.

This function creates kernel arguments from a given *hipblaslt_ext::GemmInstance* then saves the arguments inside the instance.

Parameters

- **algo** – **[in]** Handle for matrix multiplication algorithm to be used. See hipblaslt.h::hipblasLtMatmulAlgo_t . When NULL, an implicit heuristics query with default search preferences will be performed to determine actual algorithm to use.
- **workspace** – **[in]** Pointer to the workspace buffer allocated in the GPU memory. Pointer must be 16B aligned (that is, lowest 4 bits of address must be 0).
- **useUserArgs** – **[in]** Use user args, this does not affect vanilla gemm. (May be deprecated in the future)
- **stream** – **[in]** The HIP stream where all the GPU work will be submitted. (May be deprecated in the future)

Return values

- **HIPBLAS_STATUS_SUCCESS** – If the operation completed successfully.
- **HIPBLAS_STATUS_INVALID_VALUE** – If the gemm_count = 0.

hipblasStatus_t **initialize**(const *hipblasLtMatmulAlgo_t* &algo, GemmTuning &tuning, void *workspace, bool useUserArgs = true, hipStream_t stream = 0)

Create kernel arguments from a given *hipblaslt_ext::GemmInstance*.

This function creates kernel arguments from a given *hipblaslt_ext::GemmInstance* then saves the arguments inside the instance.

Parameters

- **algo** – **[in]** Handle for matrix multiplication algorithm to be used. See hipblaslt.h::hipblasLtMatmulAlgo_t . When NULL, an implicit heuristics query with default search preferences will be performed to determine actual algorithm to use.
- **tuning** – **[in]** Structure with user tuning parameters. Note that not every algo supports user tuning parameters. Will return HIPBLAS_STATUS_INVALID_VALUE if not supported. be 0).
- **workspace** – **[in]** Pointer to the workspace buffer allocated in the GPU memory. Pointer must be 16B aligned (that is, lowest 4 bits of address must be 0).
- **useUserArgs** – **[in]** Use user args, this does not affect vanilla gemm. (May be deprecated in the future)

- **stream** – **[in]** The HIP stream where all the GPU work will be submitted. (May be deprecated in the future)

Return values

- **HIPBLAS_STATUS_SUCCESS** – If the operation completed successfully.
- **HIPBLAS_STATUS_INVALID_VALUE** – If the gemm_count = 0.

hipblasStatus_t **run**(hipStream_t stream)

Execute the kernel arguments stored inside the *hipblaslt_ext::GemmInstance*.

Parameters

stream – **[in]** The HIP stream where all the GPU work will be submitted.

Return values

HIPBLAS_STATUS_SUCCESS – If the operation completed successfully.

Protected Functions

explicit **GemmInstance**(*hipblasLtHandle_t* handle, *GemmType* type)

Constructor of *GemmInstance*.

Gemm

class **Gemm** : public hipblaslt_ext::*GemmInstance*

hipblasLt extension instance for gemm.

The instance can be used to create arguments to compute the matrix multiplication of matrices A and B to produce the output matrix D, according to the following operation: `D = alpha*( A *B) + beta*( C )`, where `A`, `B`, and `C` are input matrices, and `alpha` and `beta` are input scalars.

Public Functions

explicit **Gemm**(*hipblasLtHandle_t* handle, hipblasOperation_t opA, hipblasOperation_t opB, hipDataType typeA, hipDataType typeB, hipDataType typeC, hipDataType typeD, hipblasComputeType_t typeCompute)

Constructor.

This function set the problem from hipblasLt structures. For more information about the structures, see hipblasLtMatmul for more information.

Parameters

- **handle** – **[in]** The handle from hipBLASLt.
- **opA, opB** – **[in]** The transpose type of matrix A, B
- **typeA, typeB, typeC, typeD** – **[in]** The data type of matrix A, B, C, D
- **typeCompute** – **[in]** The compute type of the gemm problem

explicit **Gemm**(*hipblasLtHandle_t* handle, *hipblasLtMatmulDesc_t* matmul_descr, const void *alpha, const void *A, *hipblasLtMatrixLayout_t* matA, const void *B, *hipblasLtMatrixLayout_t* matB, const void *beta, const void *C, *hipblasLtMatrixLayout_t* matC, void *D, *hipblasLtMatrixLayout_t* matD)

Constructor that sets the gemm problem from hipblasLt structures.

This constructor sets the problem from hipblasLt structures. For more information about the structures, see hipblasLtMatmul for more information.

Parameters

- **handle** – **[in]** The handle from hipBLASLt.
- **matmul_descr** – **[in]** Handle to a previously created matrix multiplication descriptor of type *hipblasLtMatmulDesc_t* .
- **alpha, beta** – **[in]** Pointers to the scalars used in the multiplication.
- **matA, matB, matC, matD** – **[in]** Handles to the previously created matrix layout descriptors of the type *hipblasLtMatrixLayout_t* .
- **A, B, C** – **[in]** Pointers to the GPU memory associated with the corresponding descriptors `matA`, `matB` and `matC` .
- **D** – **[out]** Pointer to the GPU memory associated with the descriptor `matD` .

hipblasStatus_t **setProblem**(int64_t m, int64_t n, int64_t k, int64_t batch_count, *GemmEpilogue* &epilogue, *GemmInputs* &inputs)

Sets the problem for a gemm problem.

This function sets the problem with m, n, k, batch_count. It uses the problem type sets from the constructor.

Parameters

- **m, n, k** – **[in]** The problem size.
- **batch_count** – **[in]** The batch count.
- **epilogue** – **[in]** The structure that controls the epilogue.
- **inputs** – **[in]** The inputs of the problem.

Return values

- **HIPBLAS_STATUS_SUCCESS** – If the operation completed successfully.
- **HIPBLAS_STATUS_EXECUTION_FAILED** – If HIP reported an execution error from the device.
- **HIPBLAS_STATUS_ARCH_MISMATCH** – If the configured operation cannot be run using the selected device.
- **HIPBLAS_STATUS_NOT_SUPPORTED** – If the current implementation on the selected device doesn't support the configured operation.
- **HIPBLAS_STATUS_INVALID_VALUE** – If the parameters are unexpectedly NULL, in conflict or in an impossible configuration.
- **HIBLAS_STATUS_NOT_INITIALIZED** – If hipBLASLt handle has not been initialized.

hipblasStatus_t **setProblem**(int64_t m, int64_t n, int64_t k, int64_t batch_count, int64_t lda, int64_t ldb, int64_t ldc, int64_t ldd, int64_t strideA, int64_t strideB, int64_t strideC, int64_t strideD, *GemmEpilogue* &epilogue, *GemmInputs* &inputs, *GemmProblemType* &problemtype)

Sets the problem for a gemm problem.

This function sets the problem with m, n, k, batch_count. It uses the problem type sets from the constructor.

Parameters

- **m, n, k** – **[in]** The problem size.
- **batch_count** – **[in]** The batch count.
- **lda, ldb, ldc, ldd** – **[in]** The leading dimensions of the matrix.
- **strideA, strideB, strideC, strideD** – **[in]** The batch stride of the matrix.
- **epilogue** – **[in]** The structure that controls the epilogue.
- **inputs** – **[in]** The inputs of the problem.
- **problemtype** – **[in]** The structure that sets the problem type of a gemm problem.

Return values

- **HIPBLAS_STATUS_SUCCESS** – If the operation completed successfully.
- **HIPBLAS_STATUS_EXECUTION_FAILED** – If HIP reported an execution error from the device.
- **HIPBLAS_STATUS_ARCH_MISMATCH** – If the configured operation cannot be run using the selected device.
- **HIPBLAS_STATUS_NOT_SUPPORTED** – If the current implementation on the selected device doesn't support the configured operation.
- **HIPBLAS_STATUS_INVALID_VALUE** – If the parameters are unexpectedly NULL, in conflict or in an impossible configuration.
- **HIBLAS_STATUS_NOT_INITIALIZED** – If hipBLASLt handle has not been initialized.

hipblasStatus_t **setProblem**(*hipblasLtMatmulDesc_t* matmul_descr, const void *alpha, const void *A, *hipblasLtMatrixLayout_t* matA, const void *B, *hipblasLtMatrixLayout_t* matB, const void *beta, const void *C, *hipblasLtMatrixLayout_t* matC, void *D, *hipblasLtMatrixLayout_t* matD)

Sets the gemm problem from hipblasLt structures.

This function sets the problem from hipblasLt structures. For more information about the structures, see hipblasLtMatmul for more information.

Parameters

- **matmul_descr** – **[in]** Handle to a previously created matrix multiplication descriptor of type *hipblasLtMatmulDesc_t* .
- **alpha, beta** – **[in]** Pointers to the scalars used in the multiplication.
- **matA, matB, matC, matD** – **[in]** Handles to the previously created matrix layout descriptors of the type *hipblasLtMatrixLayout_t* .
- **A, B, C** – **[in]** Pointers to the GPU memory associated with the corresponding descriptors `matA`, `matB` and `matC` .
- **D** – **[out]** Pointer to the GPU memory associated with the descriptor `matD` .

Return values

- **HIPBLAS_STATUS_SUCCESS** – If the operation completed successfully.
- **HIPBLAS_STATUS_EXECUTION_FAILED** – If HIP reported an execution error from the device.
- **HIPBLAS_STATUS_ARCH_MISMATCH** – If the configured operation cannot be run using the selected device.

- **HIPBLAS_STATUS_NOT_SUPPORTED** – If the current implementation on the selected device doesn't support the configured operation.
- **HIPBLAS_STATUS_INVALID_VALUE** – If the parameters are unexpectedly NULL, in conflict or in an impossible configuration.
- **HIBLAS_STATUS_NOT_INITIALIZED** – If hipBLASLt handle has not been initialized.

GroupedGemm

class **GroupedGemm** : public hipblaslt_ext::*GemmInstance*

hipblasLt extension instance for grouped gemm.

The instance can be used to create arguments to compute the matrix multiplication of matrices A and B to produce the output matrix D, according to the following operation: `D = alpha*( A *B) + beta*( C )`, where `A`, `B`, and `C` are input matrices, and `alpha` and `beta` are input scalars.

Public Functions

explicit **GroupedGemm**(*hipblasLtHandle_t* handle, hipblasOperation_t opA, hipblasOperation_t opB, hipDataType typeA, hipDataType typeB, hipDataType typeC, hipDataType typeD, hipblasComputeType_t typeCompute)

Constructor.

This function set the problem from hipblasLt structures. For more information about the structures, see hipblasLtMatmul for more information.

Parameters

- **handle** – **[in]** The handle from hipBLASLt.
- **opA, opB** – **[in]** The transpose type of matrix A, B
- **typeA, typeB, typeC, typeD** – **[in]** The data type of matrix A, B, C, D
- **typeCompute** – **[in]** The compute type of the gemm problem

explicit **GroupedGemm**(*hipblasLtHandle_t* handle, std::vector<*hipblasLtMatmulDesc_t*> &matmul_descr, std::vector<void*> &alpha, std::vector<void*> &A, std::vector<*hipblasLtMatrixLayout_t*> &matA, std::vector<void*> &B, std::vector<*hipblasLtMatrixLayout_t*> &matB, std::vector<void*> &beta, std::vector<void*> &C, std::vector<*hipblasLtMatrixLayout_t*> &matC, std::vector<void*> &D, std::vector<*hipblasLtMatrixLayout_t*> &matD)

Constructor that sets the grouped gemm problem from hipblasLt structures.

This constructor sets the problem from hipblasLt structures. For more information about the structures, see hipblasLtMatmul for more information.

Parameters

- **handle** – **[in]** The handle from hipBLASLt.
- **matmul_descr** – **[in]** Vectors of handle to a previously created matrix multiplication descriptor of type *hipblasLtMatmulDesc_t* .
- **alpha, beta** – **[in]** Vectors of float used in the multiplication.
- **matA, matB, matC, matD** – **[in]** Vectors of handle to the previously created matrix layout descriptors of the type *hipblasLtMatrixLayout_t* .

- **A, B, C** – **[in]** Vectors of pointer to the GPU memory associated with the corresponding descriptors matA, matB and matC .
- **D** – **[out]** Vector of pointer to the GPU memory associated with the descriptor matD .

hipblasStatus_t **setProblem**(std::vector<int64_t> &m, std::vector<int64_t> &n, std::vector<int64_t> &k, std::vector<int64_t> &batch_count, std::vector<*GemmEpilogue*> &epilogue, std::vector<*GemmInputs*> &inputs)

Sets the problem for a gemm problem.

This function sets the problem with m, n, k, batch_count. It uses the problem type sets from the constructor.

Parameters

- **m, n, k** – **[in]** The problem size in vector.
- **batch_count** – **[in]** The batch count in vector.
- **epilogue** – **[in]** The structure in vector that controls the epilogue.
- **inputs** – **[in]** The inputs in vector of the problem.

Return values

- **HIPBLAS_STATUS_SUCCESS** – If the operation completed successfully.
- **HIPBLAS_STATUS_EXECUTION_FAILED** – If HIP reported an execution error from the device.
- **HIPBLAS_STATUS_ARCH_MISMATCH** – If the configured operation cannot be run using the selected device.
- **HIPBLAS_STATUS_NOT_SUPPORTED** – If the current implementation on the selected device doesn't support the configured operation.
- **HIPBLAS_STATUS_INVALID_VALUE** – If the parameters are unexpectedly NULL, in conflict or in an impossible configuration.
- **HIBLAS_STATUS_NOT_INITIALIZED** – If hipBLASLt handle has not been initialized.

hipblasStatus_t **setProblem**(std::vector<int64_t> &m, std::vector<int64_t> &n, std::vector<int64_t> &k, std::vector<int64_t> &batch_count, std::vector<int64_t> &lda, std::vector<int64_t> &ldb, std::vector<int64_t> &ldc, std::vector<int64_t> &ldd, std::vector<int64_t> &strideA, std::vector<int64_t> &strideB, std::vector<int64_t> &strideC, std::vector<int64_t> &strideD, std::vector<*GemmEpilogue*> &epilogue, std::vector<*GemmInputs*> &inputs, *GemmProblemType* &problemtype)

Sets the problem for a gemm problem.

This function sets the problem with m, n, k, batch_count. It uses the problem type sets from the constructor.

Parameters

- **m, n, k** – **[in]** The problem size in vector.
- **batch_count** – **[in]** The batch count in vector.
- **lda, ldb, ldc, ldd** – **[in]** The leading dimensions in vector of the matrix.
- **strideA, strideB, strideC, strideD** – **[in]** The batch stride in vector of the matrix.
- **epilogue** – **[in]** The structure in vector that controls the epilogue.
- **inputs** – **[in]** The inputs in vector of the problem.
- **problemtype** – **[in]** The structure that sets the problem type of a gemm problem.

Return values

- **HIPBLAS_STATUS_SUCCESS** – If the operation completed successfully.
- **HIPBLAS_STATUS_EXECUTION_FAILED** – If HIP reported an execution error from the device.
- **HIPBLAS_STATUS_ARCH_MISMATCH** – If the configured operation cannot be run using the selected device.
- **HIPBLAS_STATUS_NOT_SUPPORTED** – If the current implementation on the selected device doesn't support the configured operation.
- **HIPBLAS_STATUS_INVALID_VALUE** – If the parameters are unexpectedly NULL, in conflict or in an impossible configuration.
- **HIBLAS_STATUS_NOT_INITIALIZED** – If hipBLASLt handle has not been initialized.

hipblasStatus_t **setProblem**(std::vector<*hipblasLtMatmulDesc_t*> &matmul_descr, std::vector<void*> &alpha, std::vector<void*> &A, std::vector<*hipblasLtMatrixLayout_t*> &matA, std::vector<void*> &B, std::vector<*hipblasLtMatrixLayout_t*> &matB, std::vector<void*> &beta, std::vector<void*> &C, std::vector<*hipblasLtMatrixLayout_t*> &matC, std::vector<void*> &D, std::vector<*hipblasLtMatrixLayout_t*> &matD)

Sets the grouped gemm problem from hipblasLt structures.

This function sets the problem from hipblasLt structures. For more information about the structures, see hipblasLtMatmul for more information.

Parameters

- **matmul_descr** – **[in]** Vectors of handle to a previously created matrix multiplication descriptor of type *hipblasLtMatmulDesc_t* .
- **alpha, beta** – **[in]** Vectors of float used in the multiplication.
- **matA, matB, matC, matD** – **[in]** Vectors of handle to the previously created matrix layout descriptors of the type *hipblasLtMatrixLayout_t* .
- **A, B, C** – **[in]** Vectors of pointer to the GPU memory associated with the corresponding descriptors `matA`, `matB` and `matC` .
- **D** – **[out]** Vector of pointer to the GPU memory associated with the descriptor `matD` .

Return values

- **HIPBLAS_STATUS_SUCCESS** – If the operation completed successfully.
- **HIPBLAS_STATUS_EXECUTION_FAILED** – If HIP reported an execution error from the device.
- **HIPBLAS_STATUS_ARCH_MISMATCH** – If the configured operation cannot be run using the selected device.
- **HIPBLAS_STATUS_NOT_SUPPORTED** – If the current implementation on the selected device doesn't support the configured operation.
- **HIPBLAS_STATUS_INVALID_VALUE** – If the parameters are unexpectedly NULL, in conflict or in an impossible configuration.
- **HIBLAS_STATUS_NOT_INITIALIZED** – If hipBLASLt handle has not been initialized.

hipblasStatus_t **getDefaultValueForDeviceUserArguments**(void *hostDeviceUserArgs)

A helper function to initialize DeviceUserArguments using the set problem(s) saved in the gemm object.

Parameters

hostDeviceUserArgs – **[in]** The DeviceUserArguments struture allocated in host. Note that the user must put the correct type of the DeviceUserArguments.

Return values

HIPBLAS_STATUS_SUCCESS – If the operation completed successfully.

hipblasStatus_t **run**(void *deviceUserArgs, hipStream_t stream)

Run the kernel using DeviceUserArguments.

Parameters

- **deviceUserArgs** – **[in]** Pointer to the DeviceUserArguments buffer allocated in the GPU memory. Pointer must be 16B aligned (that is, lowest 4 bits of
- **stream** – **[in]** The HIP stream where all the GPU work will be submitted.

Return values

- **HIPBLAS_STATUS_SUCCESS** – If the operation completed successfully.
- **HIPBLAS_STATUS_INVALID_VALUE** – If the gemm_count = 0.

hipblasStatus_t **run**(hipStream_t stream)

Execute the kernel arguments stored inside the *hipblaslt_ext::GemmInstance*.

Parameters

stream – **[in]** The HIP stream where all the GPU work will be submitted.

Return values

HIPBLAS_STATUS_SUCCESS – If the operation completed successfully.

5.5.3 hipBLASLtExt API Reference

getAllAlgos()

hipblasStatus_t hipblaslt_ext::**getAllAlgos**(*hipblasLtHandle_t* handle, *GemmType* typeGemm, hipblasOperation_t opA, hipblasOperation_t opB, hipDataType typeA, hipDataType typeB, hipDataType typeC, hipDataType typeD, hipblasComputeType_t typeCompute, std::vector<*hipblasLtMatmulHeuristicResult_t*> &heuristicResults)

Retrieve the possible algorithms.

This function retrieves the possible algorithms for the matrix multiply operation *hipblasLtMatmul()* function with the given data and compute tpye. The output is placed in heuristicResults in the order of increasing estimated compute time.

Parameters

- **handle** – **[in]** Pointer to the allocated hipBLASLt handle for the hipBLASLt context. See *hipblasLtHandle_t* .
- **typeGemm** – **[in]** *Gemm* type. ex. GEMM, GROUPED_GEMM.
- **opA, opB** – **[in]** Transpose settings of A, B.
- **typeA, typeB, typeC, typeD** – **[in]** The data type of matrix A, B, C, D.

- **typeCompute** – **[in]** The compute type.
- **heuristicResults** – **[out]** The algorithm heuristic vector.

Return values

- **HIPBLAS_STATUS_SUCCESS** – If query was successful. Inspect returnedAlgoCount > 0.state for the status of the results.
- **HIPBLAS_STATUS_NOT_SUPPORTED** – If no heuristic function available for current configuration.
- **HIPBLAS_STATUS_INVALID_VALUE** – If no solution is found.

getIndexFromAlgo()

int hipblaslt_ext::**getIndexFromAlgo**(*hipblasLtMatmulAlgo_t* &algo)

Retrieve the algorithm index.

Parameters

algo – **[in]** The algorithm.

Return values

int – The index of the algorithm, can be used to get heuristic results from *getAlgosFromIndex*. Returns -1 if the index stored in algo < 0. Note that the index may not be valid if the algo struct is not initialized properly.

getAlgosFromIndex()

hipblasStatus_t hipblaslt_ext::**getAlgosFromIndex**(*hipblasLtHandle_t* handle, std::vector<int> &algoIndex, std::vector<*hipblasLtMatmulHeuristicResult_t*> &heuristicResults)

Retrieve the possible algorithms.

This function retrieves the possible algorithms for the matrix multiply operation *hipblasLtMatmul()* function with the given index. The output is placed in heuristicResult in the order of increasing estimated compute time.

Parameters

- **handle** – **[in]** Pointer to the allocated hipBLASLt handle for the hipBLASLt context. See *hipblasLtHandle_t* .
- **algoIndex** – **[in]** The algorithm index vector.
- **heuristicResults** – **[out]** The algorithm heuristic vector.

Return values

- **HIPBLAS_STATUS_SUCCESS** – If query was successful. Inspect heuristicResults.size() > 0.state for the status of the results.
- **HIPBLAS_STATUS_NOT_SUPPORTED** – If no heuristic function available for current configuration.
- **HIPBLAS_STATUS_INVALID_VALUE** – If no solution is found.

matmulIsAlgoSupported()

hipblasStatus_t **hipblaslt_ext::matmulIsAlgoSupported**(*hipblasLtHandle_t* handle, *hipblasLtMatmulDesc_t* matmulDesc, const void *alpha, *hipblasLtMatrixLayout_t* Adesc, *hipblasLtMatrixLayout_t* Bdesc, const void *beta, *hipblasLtMatrixLayout_t* Cdesc, *hipblasLtMatrixLayout_t* Ddesc, *hipblasLtMatmulAlgo_t* &algo, size_t &workspaceSizeInBytes)

Check if the algorithm supports the problem. (For hipblasLt API)

This function updates the problem saved inside the algorithm if the problem is supported. The required workspaceSizeInBytes is also returned.

Parameters

- **handle** – **[in]** Pointer to the allocated hipBLASLt handle for the hipBLASLt context. See *hipblasLtHandle_t* .
- **matmulDesc** – **[in]** Handle to a previously created matrix multiplication descriptor of type *hipblasLtMatmulDesc_t* .
- **alpha, beta** – **[in]** Pointers to the scalars used in the multiplication.
- **Adesc, Bdesc, Cdesc, Ddesc** – **[in]** Handles to the previously created matrix layout descriptors of the type *hipblasLtMatrixLayout_t* .
- **algo** – **[in]** The algorithm heuristic.
- **workspaceSizeInBytes** – **[out]** Return the required workspace size.

Return values

- **HIPBLAS_STATUS_SUCCESS** – If query was successful. The problem is supported by the algorithm. results.
- **HIPBLAS_STATUS_INVALID_VALUE** – The problem is not supported.

5.5.4 hipblasLtExt Usage

Introduction

hipBLASLt has extension APIs with namespace hipblaslt_ext. It is C++ compatible only. The extensions support:

1. Gemm
2. Grouped gemm
3. Get all algorithms

Gemm

hipblasLt has its own instance.

The user must assign the problem type when construct or import the problem from hipBLAS API.

```
HIPBLASLT_EXPORT explicit Gemm(hipblasLtHandle_t      handle,
                               hipblasOperation_t     opA,
                               hipblasOperation_t     opB,
                               hipDataType      typeA,
                               hipDataType      typeB,
                               hipDataType      typeC,
                               hipDataType      typeD,
                               hipblasComputeType_t typeCompute);

HIPBLASLT_EXPORT explicit Gemm(hipblasLtHandle_t       handle,
                               hipblasLtMatmulDesc_t   matmul_descr,
                               const void*             alpha,
                               const void*             A,
                               hipblasLtMatrixLayout_t matA,
                               const void*             B,
                               hipblasLtMatrixLayout_t matB,
                               const void*             beta,
                               const void*             C,
                               hipblasLtMatrixLayout_t matC,
                               void*                   D,
                               hipblasLtMatrixLayout_t matD);
```

After the instance is created, the user can set the problem with the API. The API may requires the following structures:

GemmProblemType lets user able to change the problem type after the instance is initialized.

```
struct GemmProblemType
{
    hipblasOperation_t     op_a;
    hipblasOperation_t     op_b;
    hipDataType      type_a;
    hipDataType      type_b;
    hipDataType      type_c;
    hipDataType      type_d;
    hipblasComputeType_t type_compute;
};
```

GemmEpilogue lets user to control the epilogue of the problem.

```
struct GemmEpilogue
{
    hipblasLtEpilogue_t mode = HIPBLASLT_EPILOGUE_DEFAULT;
    hipDataType   bias_data_type;
    int                 aux_ld;
    int                 aux_stride;
};
```

GemmInputs is the problem inputs.

```
struct GemmInputs
{
    void* a = nullptr;
    void* b = nullptr;
    void* c = nullptr;
    void* d = nullptr;
    void* alpha = nullptr;
    void* beta = nullptr;
    // Epilogue inputs
    void* bias = nullptr;
    void* aux = nullptr;
};
```

And the setProblem APIs:

```
HIPBLASLT_EXPORT hipblasStatus_t setProblem(
    int64_t m, int64_t n, int64_t k, int64_t batch_count, GemmEpilogue& epilogue,␣
↪GemmInputs& inputs);
```

The user can also set the leading dimensions, strides, and reassign the data type with the following API.

```
HIPBLASLT_EXPORT hipblasStatus_t setProblem(int64_t          m,
                                            int64_t          n,
                                            int64_t          k,
                                            int64_t          batch_count,
                                            int64_t          lda,
                                            int64_t          ldb,
                                            int64_t          ldc,
                                            int64_t          ldd,
                                            int64_t          strideA,
                                            int64_t          strideB,
                                            int64_t          strideC,
                                            int64_t          strideD,
                                            GemmEpilogue&    epilogue,
                                            GemmInputs&      inputs,
                                            GemmProblemType& problemtype);
```

The user can also importing problems from hipblasLt APIs after the instance is created, note that this may overwrite the problem type of the instance.

```
HIPBLASLT_EXPORT hipblasStatus_t setProblem(hipblasLtMatmulDesc_t   matmul_descr,
                                            const void*             alpha,
                                            const void*             A,
                                            hipblasLtMatrixLayout_t matA,
                                            const void*             B,
                                            hipblasLtMatrixLayout_t matB,
                                            const void*             beta,
                                            const void*             C,
                                            hipblasLtMatrixLayout_t matC,
                                            void*                   D,
                                            hipblasLtMatrixLayout_t matD);
```

The user can get heuristic and make kernel arguments with the instance. If the properties of the gemm and the inputs don't change, the user can call the run API to launch the kernel directly.

```
// Pseudo code
hipblaslt_ext::GemmPreference pref;
pref.setMaxWorkspaceBytes(1000000);
// Default epilogue mode is HIPBLASLT_EPILOGUE_DEFAULT
hipblaslt_ext::GemmEpilogue epilogue;
hipblaslt_ext::GemmInputs inputs;
inputs.a = a;
inputs.b = b;
inputs.c = c;
inputs.d = d;
inputs.alpha = alpha;
inputs.beta = beta;

hipblaslt_ext::Gemm gemm(handle,
                         HIPBLAS_OP_N,
                         HIPBLAS_OP_N,
                         HIP_R_16F,
                         HIP_R_16F,
                         HIP_R_16F,
                         HIP_R_16F,
                         HIPBLAS_COMPUTE_32F);
std::vector<hipblasLtMatmulHeuristicResult_t> heuristic;
gemm.setProblem(1, 1, 1, 1, epilogue, inputs); // m, n, k, batch
gemm.algoGetHeuristic(gemm, pref, heuristic);
gemm.initialize(heuristic[0].algo, d_workspace, stream);
for(int i = 0; i < 10; i++)
{
    gemm.run(stream);
}
```

Grouped Gemm

hipblasLtExt supports grouped gemm. It shares the same class with normal gemm.

After the problem is set, the user can check the problem type with function getGemmType().

```
enum class GemmType
{
    HIPBLASLT_GEMM              = 1,
    HIPBLASLT_GROUPED_GEMM      = 2
};
```

The grouped gemm class also has the setProblem APIs.

```
HIPBLASLT_EXPORT hipblasStatus_t setProblem(
    int64_t m, int64_t n, int64_t k, int64_t batch_count, GemmEpilogue& epilogue,␣
↪GemmInputs& inputs);

HIPBLASLT_EXPORT hipblasStatus_t setProblem(std::vector<int64_t>&      m,
                                            std::vector<int64_t>&      n,
                                            std::vector<int64_t>&      k,
                                            std::vector<int64_t>&      batch_count,
```

(continues on next page)

(continued from previous page)

```
                                        std::vector<GemmEpilogue>& epilogue,
                                        std::vector<GemmInputs>&   inputs);

HIPBLASLT_EXPORT hipblasStatus_t setProblem(std::vector<int64_t>&      m,
                                        std::vector<int64_t>&      n,
                                        std::vector<int64_t>&      k,
                                        std::vector<int64_t>&      batch_count,
                                        std::vector<int64_t>&      lda,
                                        std::vector<int64_t>&      ldb,
                                        std::vector<int64_t>&      ldc,
                                        std::vector<int64_t>&      ldd,
                                        std::vector<int64_t>&      strideA,
                                        std::vector<int64_t>&      strideB,
                                        std::vector<int64_t>&      strideC,
                                        std::vector<int64_t>&      strideD,
                                        std::vector<GemmEpilogue>& epilogue,
                                        std::vector<GemmInputs>&   inputs,
                                        GemmProblemType&           problemtype);

HIPBLASLT_EXPORT hipblasStatus_t setProblem(std::vector<hipblasLtMatmulDesc_t>&   matmul_
↪descr,
                                        std::vector<void*>&                    alpha,
                                        std::vector<void*>&                    A,
                                        std::vector<hipblasLtMatrixLayout_t>& matA,
                                        std::vector<void*>&                    B,
                                        std::vector<hipblasLtMatrixLayout_t>& matB,
                                        std::vector<void*>&                    beta,
                                        std::vector<void*>&                    C,
                                        std::vector<hipblasLtMatrixLayout_t>& matC,
                                        std::vector<void*>&                    D,
                                        std::vector<hipblasLtMatrixLayout_t>& matD);
```

For the following API, the argument "epilogue" supports broadcasting. They will be broadcasted to the length of the problem size by duplicating the last element.

```
HIPBLASLT_EXPORT hipblasStatus_t setProblem(std::vector<int64_t>&      m,
                                        std::vector<int64_t>&      n,
                                        std::vector<int64_t>&      k,
                                        std::vector<int64_t>&      batch_count,
                                        std::vector<int64_t>&      lda,
                                        std::vector<int64_t>&      ldb,
                                        std::vector<int64_t>&      ldc,
                                        std::vector<int64_t>&      ldd,
                                        std::vector<int64_t>&      strideA,
                                        std::vector<int64_t>&      strideB,
                                        std::vector<int64_t>&      strideC,
                                        std::vector<int64_t>&      strideD,
                                        std::vector<GemmEpilogue>& epilogue,
                                        std::vector<GemmInputs>&   inputs,
                                        GemmProblemType&           problemtype);
```

Note that currently only supports problemtype size equals to 1 (Only one GemmProblemType for all problems).

```
// Pseudo code
std::vector<int64_t> m, n, k;
// ...
for(size_t i = 0; i < problem_size, i++)
{
    // ...
}
std::vector<GemmProblemType> problemtypes;
problemtypes.push_back(problemtype);
groupedgemm.setProblem(m, n, k, batch_count, lda, ldb, ldc, ldd, strideA, strideB,␣
↪strideC, strideD, epilogue, inputs, problemtypes);
```

UserArguments

Grouped gemm supports using external device memory to run the kernel. This will be helpful if some of the arguments are from the output of the pervious kernel. Please refer to section Fixed MK if you want to change the size (m, n, k, batch) related arguments.

```
struct UserArguments
{
    uint32_t m; //!< size m
    uint32_t n; //!< size n
    uint32_t batch; //!< size batch
    uint32_t k; //!< size k
    void*    d; //!< The d matrix input pointer.
    void*    c; //!< The c matrix input pointer.
    void*    a; //!< The a matrix input pointer.
    void*    b; //!< The b matrix input pointer.
    uint32_t strideD1; //!< The d leading dimension.
    uint32_t strideD2; //!< The d batch stride
    uint32_t strideC1; //!< The c leading dimension.
    uint32_t strideC2; //!< The c batch stride
    uint32_t strideA1; //!< The a leading dimension.
    uint32_t strideA2; //!< The a batch stride
    uint32_t strideB1; //!< The b leading dimension.
    uint32_t strideB2; //!< The b batch stride
    int8_t   alpha[16]; //!< The alpha value.
    int8_t   beta[16]; //!< The beta value.
    // Epilogue inputs
    void*    bias; //!< The bias input pointer.
    int      biasType; //!< The bias datatype. Only works if mode is set to bias related␣
↪epilogues.
    uint32_t reserved;
    void*    e; //!< The aux input pointer. Only works if mode is set to aux related␣
↪epilogues.
    uint32_t strideE1; //!< The aux leading dimension. Only works if mode is set to aux␣
↪related epilogues.
    uint32_t strideE2; //!< The aux batch stride. Only works if mode is set to aux␣
↪related epilogues.
    float    act0; //!< The activation value 1. Some activations might use it.
    float    act1; //!< The activation value 2.
```

(continues on next page)

(continued from previous page)

```
    int      activationType; //!< The activation type.  Only works if mode is set to␣
↪activation related epilogues.
} __attribute__((packed));
```

We add the two functions for UserArguments related API. The first API is a helper function that helps the user to initialize the structure "UserArguments" from the saved problems inside the grouped gemm object. The second API is an overload function with an additional UserArguments device pointer input.

```
HIPBLASLT_EXPORT hipblasStatus_t getDefaultValueForDeviceUserArguments(void*␣
↪hostDeviceUserArgs);

HIPBLASLT_EXPORT hipblasStatus_t run(void* deviceUserArgs, hipStream_t stream);
```

The following is a simple example of how this API works.

```
// Pseudo code
// Step 1: Get all algorithms
std::vector<hipblasLtMatmulHeuristicResult_t> heuristicResult;
CHECK_HIPBLASLT_ERROR(hipblaslt_ext::getAllAlgos(handle,
                                                 HIPBLASLT_GEMM,
                                                 HIPBLAS_OP_N,
                                                 HIPBLAS_OP_N,
                                                 in_out_datatype,
                                                 in_out_datatype,
                                                 in_out_datatype,
                                                 in_out_datatype,
                                                 HIPBLAS_COMPUTE_32F,
                                                 heuristicResult));

hipblaslt_ext::GemmPreference pref;
pref.setMaxWorkspaceBytes(1000000);
// Step 2: Setup problem
std::vector<int64_t> m(gemm_count);
std::vector<int64_t> n(gemm_count);
std::vector<int64_t> k(gemm_count);
std::vector<int64_t> batch_count(gemm_count);
std::vector<hipblaslt_ext::GemmEpilogue> epilogue(gemm_count);
std::vector<hipblaslt_ext::GemmInputs> inputs(gemm_count);
for(int i = 0; i < gemm_count; i++)
{
    m[i] = 1;
    n[i] = 1;
    k[i] = 1;
    batch_count[i] = 1;
    epilogue[i].mode = HIPBLASLT_EPILOGUE_GELU;
    inputs[i].a = a[i];
    inputs[i].b = b[i];
    inputs[i].c = c[i];
    inputs[i].d = d[i];
    inputs[i].alpha = alpha[i];
    inputs[i].beta = beta[i];
}
```

(continues on next page)

(continued from previous page)

```
// Step 3: Create grouped gemm instance
hipblaslt_ext::GroupedGemm groupedGemm(handle,
                                       HIPBLAS_OP_N,
                                       HIPBLAS_OP_N,
                                       HIP_R_16F,
                                       HIP_R_16F,
                                       HIP_R_16F,
                                       HIP_R_16F,
                                       HIPBLAS_COMPUTE_32F);

// Step 4: Set problem
groupedGemm.setProblem(m, n, k, batch_count, epilogue, inputs); // m, n, k, batch

// Step 5: Get default value from the instance
hipblaslt_ext::UserArguments* dUAFloat = new hipblaslt_ext::UserArguments[gemm_count];
groupedGemm.getDefaultValueForDeviceUserArguments((void*)dUAFloat);
// Once you get the default value here, you can make several copies and change the values
// from the host

// Next Copy them to the device memory
hipblaslt_ext::UserArguments* d_dUAFloat = nullptr;
hipMalloc(&d_dUAFloat, sizeof(hipblaslt_ext::UserArguments) * gemm_count);
hipMemcpy(d_dUAFloat, dUAFloat, sizeof(hipblaslt_ext::UserArguments) * gemm_count,␣
↪hipMemcpyHostToDevice);

validIdx.clear();
for(int j = 0; j < heuristicResult.size(); j++)
{
    size_t workspace_size = 0;
    if(groupedGemm.isAlgoSupported(heuristicResult[j].algo, workspace_size)
       == HIPBLAS_STATUS_SUCCESS)
    {
        validIdx.push_back(j);
    }
}

// Step 6: Initialize and run
if(validIdx.size() > 1)
{
    groupedGemm.initialize(heuristicResult[validIdx[0]].algo, d_workspace, stream);
    for(int i = 0; i < 10; i++)
    {
        groupedGemm.run(userArgs, stream);
    }
}
```

The base class (GemmInstance)

This is the base class of class Gemm and GroupedGemm.

```
// Gets huesristic from the instance.
HIPBLASLT_EXPORT hipblasStatus_t algoGetHeuristic(const int                                 ␣
↪        requestedAlgoCount,
                                                  const GemmPreference&                     ␣
↪        pref,
                                                  std::vector
↪<hipblasLtMatmulHeuristicResult_t>& heuristicResults);

// Returns SUCCESS if the algo is supported, also returns the required workspace size in␣
↪bytes.
HIPBLASLT_EXPORT hipblasStatus_t isAlgoSupported(hipblasLtMatmulAlgo_t& algo, size_t&␣
↪workspaceSizeInBytes);

// Initializes the instance before calling run. Requires every time the problem is set.
HIPBLASLT_EXPORT hipblasStatus_t initialize(const hipblasLtMatmulAlgo_t& algo, void*␣
↪workspace, bool useUserArgs = true, hipStream_t stream = 0);

// Run the problem.
HIPBLASLT_EXPORT hipblasStatus_t run(hipStream_t stream);
```

Get all algorithms

Get all algorithms lets users to get all the algorithms of a specific problem type. It requires the transpose of A, B, the data type of the inputs, and the compute type.

```
HIPBLASLT_EXPORT
hipblasStatus_t hipblaslt_ext::getAllAlgos(hipblasLtHandle_t                                ␣
↪ handle,
                                           hipblasLtExtGemmTypeEnum_t                       ␣
↪ typeGemm,
                                           hipblasOperation_t                               ␣
↪ opA,
                                           hipblasOperation_t                               ␣
↪ opB,
                                           hipDataType                                 ␣
↪typeA,
                                           hipDataType                                 ␣
↪typeB,
                                           hipDataType                                 ␣
↪typeC,
                                           hipDataType                                 ␣
↪typeD,
                                           hipblasComputeType_t                             ␣
↪typeCompute,
                                           std::vector<hipblasLtMatmulHeuristicResult_t>&
↪ heuristicResults);
```

This API does not require any problem size or epilogue as input, but will use another API “isAlgoSupported” to check if the algorithm supports a problem.

```
hipblaslt_ext::matmulIsAlgoSupported()
gemm.isAlgoSupported()
```

The API will return the required workspace size in bytes if success.

```
// Get all algorithms
CHECK_HIPBLASLT_ERROR(hipblaslt_ext::getAllAlgos(handle,
                                                 HIPBLASLT_GEMM,
                                                 HIPBLAS_OP_N,
                                                 HIPBLAS_OP_N,
                                                 in_out_datatype,
                                                 in_out_datatype,
                                                 in_out_datatype,
                                                 in_out_datatype,
                                                 HIPBLAS_COMPUTE_32F,
                                                 heuristicResult));

validIdx.clear();
for(int j = 0; j < heuristicResult.size(); j++)
{
    size_t workspace_size = 0;
    if(hipblaslt_ext::matmulIsAlgoSupported(handle,
                                            matmul,
                                            &(alpha),
                                            matA,
                                            matB,
                                            &(beta),
                                            matC,
                                            matD,
                                            heuristicResult[j].algo,
                                            workspace_size)
       == HIPBLAS_STATUS_SUCCESS)
    {
        validIdx.push_back(j);
        heuristicResult[j].workspaceSize = workspace_size;
    }
    else
    {
        heuristicResult[j].workspaceSize = 0;
    }
}
```

Using extension APIs.

Gemm

```
// Pseudo code for gemm problem
// Get all algorithms
std::vector<hipblasLtMatmulHeuristicResult_t> heuristicResult;
CHECK_HIPBLASLT_ERROR(hipblaslt_ext::getAllAlgos(handle,
                                                 HIPBLASLT_GEMM,
                                                 HIPBLAS_OP_N,
                                                 HIPBLAS_OP_N,
                                                 in_out_datatype,
                                                 in_out_datatype,
                                                 in_out_datatype,
                                                 in_out_datatype,
                                                 HIPBLAS_COMPUTE_32F,
                                                 heuristicResult));

hipblaslt_ext::GemmPreference pref;
pref.setMaxWorkspaceBytes(1000000);
hipblaslt_ext::GemmEpilogue epilogue;
epilogue.mode = HIPBLASLT_EPILOGUE_GELU;
hipblaslt_ext::GemmInputs inputs;
inputs.a = a;
inputs.b = b;
inputs.c = c;
inputs.d = d;
inputs.alpha = alpha;
inputs.beta = beta;

hipblaslt_ext::Gemm gemm(handle,
                         HIPBLAS_OP_N,
                         HIPBLAS_OP_N,
                         HIP_R_16F,
                         HIP_R_16F,
                         HIP_R_16F,
                         HIP_R_16F,
                         HIPBLAS_COMPUTE_32F);

gemm.setProblem(1, 1, 1, 1, epilogue, inputs); // m, n, k, batch

validIdx.clear();
for(int j = 0; j < heuristicResult.size(); j++)
{
    size_t workspace_size = 0;
    if(gemm.isAlgoSupported(heuristicResult[j].algo, workspace_size)
       == HIPBLAS_STATUS_SUCCESS)
    {
        validIdx.push_back(j);
        heuristicResult[j].workspaceSize = workspace_size;
    }
    else
    {
        heuristicResult[j].workspaceSize = 0;
```

(continues on next page)

(continued from previous page)

```
    }
}

if(validIdx.size() > 1)
{
    gemm.initialize(heuristicResult[validIdx[0]].algo, d_workspace, stream);
    for(int i = 0; i < 10; i++)
    {
        gemm.run(stream);
    }
}
```

Grouped gemm

```
// Pseudo code for grouped gemm problem
// Get all algorithms
std::vector<hipblasLtMatmulHeuristicResult_t> heuristicResult;
CHECK_HIPBLASLT_ERROR(hipblaslt_ext::getAllAlgos(handle,
                                                 HIPBLASLT_GEMM,
                                                 HIPBLAS_OP_N,
                                                 HIPBLAS_OP_N,
                                                 in_out_datatype,
                                                 in_out_datatype,
                                                 in_out_datatype,
                                                 in_out_datatype,
                                                 HIPBLAS_COMPUTE_32F,
                                                 heuristicResult));

hipblaslt_ext::GemmPreference pref;
pref.setMaxWorkspaceBytes(1000000);

std::vector<int64_t> m(gemm_count);
std::vector<int64_t> n(gemm_count);
std::vector<int64_t> k(gemm_count);
std::vector<int64_t> batch_count(gemm_count);
std::vector<hipblaslt_ext::GemmEpilogue> epilogue(gemm_count);
std::vector<hipblaslt_ext::GemmInputs> inputs(gemm_count);
for(int i = 0; i < gemm_count; i++)
{
    m[i] = 1;
    n[i] = 1;
    k[i] = 1;
    batch_count[i] = 1;
    epilogue[i].mode = HIPBLASLT_EPILOGUE_GELU;
    inputs[i].a = a[i];
    inputs[i].b = b[i];
    inputs[i].c = c[i];
    inputs[i].d = d[i];
    inputs[i].alpha = alpha[i];
    inputs[i].beta = beta[i];
```

(continues on next page)

(continued from previous page)

```
}


hipblaslt_ext::GroupedGemm groupedGemm(handle,
                                       HIPBLAS_OP_N,
                                       HIPBLAS_OP_N,
                                       HIP_R_16F,
                                       HIP_R_16F,
                                       HIP_R_16F,
                                       HIP_R_16F,
                                       HIPBLAS_COMPUTE_32F);

groupedGemm.setProblem(m, n, k, batch_count, epilogue, inputs); // m, n, k, batch

validIdx.clear();
for(int j = 0; j < heuristicResult.size(); j++)
{
    size_t workspace_size = 0;
    if(groupedGemm.isAlgoSupported(heuristicResult[j].algo, workspace_size)
       == HIPBLAS_STATUS_SUCCESS)
    {
        validIdx.push_back(j);
    }
}

if(validIdx.size() > 1)
{
    groupedGemm.initialize(heuristicResult[validIdx[0]].algo, d_workspace, stream);
    for(int i = 0; i < 10; i++)
    {
        groupedGemm.run(stream);
    }
}
```

Algorithm Index

The extension API lets user to get the algorithm index from hipblasLtMatmulAlgo_t.

```
HIPBLASLT_EXPORT int getIndexFromAlgo(hipblasLtMatmulAlgo_t& algo);
```

It also supports user to get the heuristic results by giving an index vector.

```
HIPBLASLT_EXPORT
hipblasStatus_t
    getAlgosFromIndex(hipblasLtHandle_t                              handle,
                      std::vector<int>&                              algoIndex,
                      std::vector<hipblasLtMatmulHeuristicResult_t>& heuristicResults);
```

Example code

[Grouped Gemm] Fixed MK

hipBLASLt extension supports changing the sizes (m, n, k, batch) from the device memory "UserArguments", but the setup is a bit different from the normal routing.

Sum of n

A sum of N is required to use as an input for the grouped gemm instance.

For example, we have a grouped gemm with gemm_count = 4. The sum of N must not exceed the "sum of N" set in setProblem API. In this mode, the first element is the "sum of n" in the array of Ns.

```
// Pseudo code
// Step 1: Get all algorithms
std::vector<hipblasLtMatmulHeuristicResult_t> heuristicResult;
CHECK_HIPBLASLT_ERROR(hipblaslt_ext::getAllAlgos(handle,
                                                 HIPBLASLT_GEMM,
                                                 HIPBLAS_OP_N,
                                                 HIPBLAS_OP_N,
                                                 in_out_datatype,
                                                 in_out_datatype,
                                                 in_out_datatype,
                                                 in_out_datatype,
                                                 HIPBLAS_COMPUTE_32F,
                                                 heuristicResult));

hipblaslt_ext::GemmPreference pref;
pref.setMaxWorkspaceBytes(1000000);
// Step 2: Setup problem
std::vector<int64_t> m(gemm_count);
std::vector<int64_t> n(gemm_count);
std::vector<int64_t> k(gemm_count);
std::vector<int64_t> batch_count(gemm_count);
std::vector<hipblaslt_ext::GemmEpilogue> epilogue(gemm_count);
std::vector<hipblaslt_ext::GemmInputs> inputs(gemm_count);

// Step 2.1: Calculate sum of n
int64_t sum_of_n = 0;
for(int i = 0; i < gemm_count; i++)
{
    sum_of_n += n_arr[i];
}

// {sum_of_n, 1, 1, 1, ...}; // The array of N, the first element is the sum of N
for(int i = 0; i < gemm_count; i++)
{
    m[i] = m_arr[i];
    if(i == 0)
        n[i] = sum_of_n;
    else
```

(continues on next page)

(continued from previous page)

```
        n[i] = 1;
    k[i] = k_arr[i];
    batch_count[i] = 1;
    inputs[i].a = a[i];
    inputs[i].b = b[i];
    inputs[i].c = c[i];
    inputs[i].d = d[i];
    inputs[i].alpha = alpha[i];
    inputs[i].beta = beta[i];
}

// Step 3: Create grouped gemm instance
hipblaslt_ext::GroupedGemm groupedGemm(handle,
                                       HIPBLAS_OP_N,
                                       HIPBLAS_OP_N,
                                       HIP_R_16F,
                                       HIP_R_16F,
                                       HIP_R_16F,
                                       HIP_R_16F,
                                       HIPBLAS_COMPUTE_32F);

// Step 4: Set problem
groupedGemm.setProblem(m, n, k, batch_count, epilogue, inputs); // m, n, k, batch

// Step 5: Get default value from the instance
hipblaslt_ext::UserArguments* dUAFloat = new hipblaslt_ext::UserArguments[gemm_count];
groupedGemm.getDefaultValueForDeviceUserArguments((void*)dUAFloat);
// Once you get the default value here, you can make several copies and change the values
// from the host

// Next Copy them to the device memory
hipblaslt_ext::UserArguments* d_dUAFloat = nullptr;
hipMalloc(&d_dUAFloat, sizeof(hipblaslt_ext::UserArguments) * gemm_count);
hipMemcpy(d_dUAFloat, dUAFloat, sizeof(hipblaslt_ext::UserArguments) * gemm_count,␣
↪hipMemcpyHostToDevice);

validIdx.clear();
for(int j = 0; j < heuristicResult.size(); j++)
{
    size_t workspace_size = 0;
    if(groupedGemm.isAlgoSupported(heuristicResult[j].algo, workspace_size)
       == HIPBLAS_STATUS_SUCCESS)
    {
        validIdx.push_back(j);
    }
}

int threads = 256;
int blocks  = ceil((double)gemm_count / threads);

// Step 6: Initialize and run
if(validIdx.size() > 1)
```

(continues on next page)

(continued from previous page)

```
{
    groupedGemm.initialize(heuristicResult[validIdx[0]].algo, d_workspace);
    for(int i = 0; i < 10; i++)
    {
        hipLaunchKernelGGL(kernelUpdateN,
                           dim3(blocks),
                           dim3(threads),
                           0,
                           stream,
                           gemm_count,
                           d_dUAFloat,
                           d_n_vec);  // d_n_vec is a device pointer with Ns
        groupedGemm.run(userArgs, stream);
    }
}

// .....

__global__ void kernelUpdateN(uint32_t gemm_count, void* userArgs, int32_t* sizes_n)
{
uint64_t id = hipBlockIdx_x * 256 + hipThreadIdx_x;

if(id >= gemm_count)
    return;

hipblaslt_ext::UserArguments* dUAFloat = static_cast<hipblaslt_ext::UserArguments*>
↪(userArgs);
dUAFloat[id].n                         = sizes_n[id];
}
```

5.6 hipBLASLtExt Operation Reference

5.6.1 hipBLASLtExt Operation API Reference

hipblasltExtSoftmax()

hipblasStatus_t **hipblasltExtSoftmax**(hipDataType datatype, uint32_t m, uint32_t n, uint32_t dim, void *output, void *input, hipStream_t stream)

Perform softmax on given tensor.

This function computes softmax on given 2D-tensor along specified dimension.

Parameters

- **datatype** – **[in]** Datatype of input/output tensor, currently support HIP_R_32F only.
- **m** – **[in]** The first dimension of input/output tensor.
- **n** – **[in]** The second dimension of input/output tensor. Currently only values less than or equal to 256 are supported.
- **dim** – **[in]** Specified dimension to perform softmax on. Currently 1 is the only valid value.

- **input** – **[in]** input tensor buffer.
- **stream** – **[in]** The HIP stream where all the GPU work will be submitted.
- **output** – **[out]** output tensor buffer.

Return values

- **HIPBLAS_STATUS_SUCCESS** – If it runs successfully.
- **HIPBLAS_STATUS_INVALID_VALUE** – If `n` is greater than 256.
- **HIPBLAS_STATUS_NOT_SUPPORTED** – If `dim` is not 1 or `datatype` is not HIP_R_32F.

hipblasltExtLayerNorm()

hipblasStatus_t **hipblasltExtLayerNorm**(hipDataType datatype, void *output, void *mean, void *invvar, void *input, uint32_t m, uint32_t n, float eps, void *gamma, void *beta, hipStream_t stream)

Perform 2-D layernorm on with source input tensor and result output tensor.

This function computes layernorm on given 2D-tensor.

Parameters

- **datatype** – **[in]** Datatype of input/output tensor, currently support HIP_R_32F only.
- **output** – **[out]** output tensor buffer. can't be nullptr.
- **mean** – **[out]** tensor buffer. can't be nullptr.
- **invvar** – **[out]** tensor buffer. 1 / sqrt(std). can't be nullptr.
- **input** – **[in]** tensor buffer. can't be nullptr.
- **m** – **[in]** The first dimension of input/output tensor.
- **n** – **[in]** The second dimension of input/output tensor.
- **eps** – **[in]** for sqrt to avoid inf value.
- **gamma** – **[in]** tensor buffer. nullptr means calculation doesn't involve gamma.
- **beta** – **[in]** tensor buffer. nullptr means calculation doesn't involve beta.
- **stream** – **[in]** The HIP stream where all the GPU work will be submitted.

Return values

- **HIPBLAS_STATUS_SUCCESS** – If it runs successfully.
- **HIPBLAS_STATUS_INVALID_VALUE** – If `m` is greater than 4096.
- **HIPBLAS_STATUS_NOT_SUPPORTED** – if `datatype` is not HIP_R_32F.

hipblasltExtAMax()

hipblasStatus_t **hipblasltExtAMax**(const hipDataType datatype, const hipDataType outDatatype, void *output, void *input, uint32_t m, uint32_t n, hipStream_t stream)

Perform absmax on given 2-D tensor and output one value absmax(tensor) value.

This function computes amax on given 1D-tensor.

Parameters

- **datatype** – **[in]** Datatype of input/output tensor, currently support HIP_R_32F and HIP_R_16F only.
- **outDatatype** – **[in]** Datatype of input/output tensor, currently support HIP_R_32F and HIP_R_16F only.
- **output** – **[out]** output tensor buffer. can't be nullptr.
- **input** – **[in]** tensor buffer. can't be nullptr.
- **m** – **[in]** The first dimension of input/output tensor.
- **n** – **[in]** The first dimension of input/output tensor.
- **stream** – **[in]** The HIP stream where all the GPU work will be submitted.

Return values

- **HIPBLAS_STATUS_SUCCESS** – If it runs successfully.
- **HIPBLAS_STATUS_INVALID_VALUE** – If `m` is greater than 4096.
- **HIPBLAS_STATUS_NOT_SUPPORTED** – if `datatype` is not (HIP_R_32F or HIP_R_16F).

5.6.2 hipblasLtExt Operation Usage

Introduction

hipBLASLt has extension operation APIs which is independent to gemm operation with. These extensions support:

1. **hipblasltExtSoftmax**

   Softmax for 2D-tensor. Currently it performs softmax on second dimension of input tensor and it assumes input is contigious on second dimension.
   For sample usage, please refer to clients/benchmarks/client_ext_op_softmax.cpp

2. **hipblasltExtLayerNorm**

   Convert a 2D tensor by LayerNorm to generate a new 2D normalized tensor.
   it is a independent function which can just call and get result.
   sample code is in clients/samples/ext_op/sample_hipblaslt_ext_op_layernorm.cpp

3. **hipblasltExtAMax**

   Abs Maximum value of a 2D tensor.
   it is a independent function which can just call and get result.
   sample code is in clients/samples/ext_op/sample_hipblaslt_ext_op_amax.cpp

5.7 Clients

There are 2 main client executables that can be used with hipBLASLt.

1. hipblaslt-test
2. hipblaslt-bench

These clients can be built by following the instructions in the Build and Install hipBLASLt github page . After building the hipBLASLt clients, they can be found in the directory `hipBLASLt/build/release/clients/staging`. The next section will cover a brief explanation and the usage of each hipBLASLt clients.

5.7.1 hipblaslt-test

hipblaslt-test is the main regression gtest for hipBLASLt. All test items should pass.

Run full test items:

```
./hipblaslt-test
```

Run partial test items with filter:

```
./hipblaslt-test --gtest_filter=<test pattern>
```

Demo "quick" test:

```
./hipblaslt-test --gtest_filter=*quick*
```

5.7.2 hipblaslt-bench

hipblaslt-bench is used to measure performance and to verify the correctness of hipBLASLt functions.

It has a command line interface.

For example, run fp32 GEMM with validation:

```
./hipblaslt-bench --precision f32_r -v
transA,transB,M,N,K,alpha,lda,stride_a,beta,ldb,stride_b,ldc,stride_c,ldd,stride_d,d_
↪type,compute_type,activation_type,bias_vector,hipblaslt-Gflops,us
N,N,128,128,128,1,128,16384,0,128,16384,128,16384,128,16384,f32_r,f32_r,none,0, 415.278,␣
↪10.
```

For more information:

```
./hipblaslt-bench --help
--sizem |-m <value>        Specific matrix size: the number of rows or columns in matrix.
↪                      (Default value is: 128)
--sizen |-n <value>        Specific matrix the number of rows or columns in matrix      ␣
↪                      (Default value is: 128)
--sizek |-k <value>        Specific matrix size: the number of columns in A and rows in␣
↪B.                     (Default value is: 128)
--lda <value>              Leading dimension of matrix A.
--ldb <value>              Leading dimension of matrix B.
--ldc <value>              Leading dimension of matrix C.
```

(continues on next page)

(continued from previous page)

```
--ldd <value>                Leading dimension of matrix D.
--lde <value>                Leading dimension of matrix E.
--any_stride                 Do not modify input strides based on leading dimensions
--stride_a <value>           Specific stride of strided_batched matrix A, second dimension␣
↪* leading dimension.
--stride_b <value>           Specific stride of strided_batched matrix B, second dimension␣
↪* leading dimension.
--stride_c <value>           Specific stride of strided_batched matrix C, second dimension␣
↪* leading dimension.
--stride_d <value>           Specific stride of strided_batched matrix D, second dimension␣
↪* leading dimension.
--stride_e <value>           Specific stride of strided_batched matrix E, second dimension␣
↪* leading dimension.
--alpha <value>              specifies the scalar alpha                                    ␣
↪                          (Default value is: 1)
--beta <value>               specifies the scalar beta                                     ␣
↪                          (Default value is: 0)
--function |-f <value>       BLASLt function to test. Options: matmul                      ␣
↪                          (Default value is: matmul)
--precision |-r <value>      Precision of matrix A,B,C,D  Options: f32_r,f16_r,bf16_r,f64_
↪r,i32_r,i8_r               (Default value is: f16_r)
--a_type <value>             Precision of matrix A. Options: f32_r,f16_r,bf16_r
--b_type <value>             Precision of matrix B. Options: f32_r,f16_r,bf16_r
--c_type <value>             Precision of matrix C. Options: f32_r,f16_r,bf16_r
--d_type <value>             Precision of matrix D. Options: f32_r,f16_r,bf16_r
--compute_type <value>       Precision of computation. Options: s,f32_r,x,xf32_r,f64_r,i32_
↪r                          (Default value is: f32_r)
--scale_type <value>         Precision of scalar. Options: f16_r,bf16_r
--initialization <value>     Initialize matrix data.Options: rand_int, trig_float,␣
↪hpl(floating)                     (Default value is: hpl)
--transA <value>             N = no transpose, T = transpose, C = conjugate transpose      ␣
↪                          (Default value is: N)
--transB <value>             N = no transpose, T = transpose, C = conjugate transpose      ␣
↪                          (Default value is: N)
--batch_count <value>        Number of matrices. Only applicable to batched and strided_
↪batched routines            (Default value is: 1)
--HMM                        Parameter requesting the use of HipManagedMemory
--verify |-v                 Validate GPU results with CPU?
--iters |-i <value>          Iterations to run inside timing loop                          ␣
↪                          (Default value is: 10)
--cold_iters |-j <value>     Cold Iterations to run before entering the timing loop        ␣
↪                          (Default value is: 2)
--algo_method <value>        Use different algorithm search API. Options: heuristic, all,␣
↪index.                     (Default value is: heuristic)
--solution_index <value>     Used with --algo_method 2.  Specify solution index to use in␣
↪benchmark.                 (Default value is: -1)
--requested_solution <value> Requested solution num. Set to -1 to get all solutions.␣
↪Only valid when algo_method is set to heuristic.  (Default value is: 1)
--activation_type <value>  Options: None, gelu, relu                                       ␣
↪                          (Default value is: none)
--activation_arg1 <value>  Reserved.                                                       ␣
↪                          (Default value is: 0)
```

(continues on next page)

(continued from previous page)

```
--activation_arg2 <value>  Reserved.                                                ␣
↪                       (Default value is: inf)
--bias_type <value>        Precision of bias vector.Options: f16_r,bf16_r,f32_r,
↪default(same with D type)
--bias_source <value>      Choose bias source: a, b, d                              ␣
↪                       (Default value is: d)
--bias_vector              Apply bias vector
--scaleA                   Apply scale for A buffer
--scaleB                   Apply scale for B buffer
--scaleAlpha_vector        Apply scaleAlpha vector
--amaxScaleA               Apple scale for A buffer by abs max of A buffer
--amaxScaleB               Apple scale for B buffer by abs max of B buffer
--use_e                    Apply AUX output/ gradient input
--gradient                 Enable gradient
--grouped_gemm             Use grouped_gemm.
--use_user_args            Use UserArguments located in device memory for grouped gemm.
--device <value>           Set default device to be used for subsequent program runs    ␣
↪                       (Default value is: 0)
--c_noalias_d              C and D are stored in separate memory
--workspace <value>        Set fixed workspace memory size instead of using hipblaslt␣
↪managed memory            (Default value is: 0)
--log_function_name        Function name precedes other items.
--function_filter <value>  Simple strstr filter on function name only without wildcards
--api_method <value>       Use extension API. c: C style API. mix: declaration with C␣
↪hipblasLtMatmul Layout/Desc but set, initialize, and run the problem with C++␣
↪extension API. cpp: Using C++ extension API only. Options: c, mix, cpp.  (Default␣
↪value is: c)
--print_kernel_info        Print solution, kernel name and solution index.
--rotating <value>         Use rotating memory blocks for each iteration, size in MB.   ␣
↪                       (Default value is: 0)
--use_gpu_timer            Use hipEventElapsedTime to profile elapsed time.             ␣
↪                       (Default value is: false)
--splitk <value>           [Tuning parameter] Set split K for a solution, 0 is use␣
↪solution's default value. (Only support GEMM + api_method mix or cpp)
--help |-h                 produces this help message
--version <value>          Prints the version number
```

INDEX

H