hipSPARSELt Data Types

Data Structures

`hipsparseLtHandle_t`

The structure holds the hipSPARSELt library context (device properties, system information, etc.).
The handle must be initialized and destroyed with hipsparseLtInit() and hipsparseLtDestroy() functions respectively.

`hipsparseLtMatDescriptor_t`

The structure captures the shape and characteristics of a matrix.
It is initialized with hipsparseLtDenseDescriptorInit() or hipsparseLtStructuredDescriptorInitfunctions and destroyed with hipsparseLtMatDescriptorDestroy().

`hipsparseLtMatDescriptor_t`

The structure holds the description of the matrix multiplication operation.
It is initialized with hipsparseLtMatmulDescriptorInit() function.

`hipsparseLtMatmulAlgSelection_t`

The structure holds the description of the matrix multiplication algorithm.
It is initialized with hipsparseLtMatmulAlgSelectionInit() function.

`hipsparseLtMatmulPlan_t`

The structure holds the matrix multiplication execution plan, namely all the information necessary to execute the `hipsparseLtMatmull()` operation.
It is initialized and destroyed with hipsparseLtMatmulPlanInit() and hipsparseLtMatmulPlanDestroy() functions respectively.

Enumerators

`hipsparseLtSparsity_t`

The enumerator specifies the sparsity ratio of the structured matrix as

$$sparisty\ ratio = \frac{nnz}{num\_rows * num\_cols}$$

Value	Description
`HIPSPARSELT_SPARSITY_50_PERCENT`	50% Sparsity Ratio: - **2:4** for `half`, `bfloat16`, `int8` - 1:2 for `tf32`, `int` (CUDA only)

The sparsity property is used in the hipsparseLtStructuredDescriptorInit() function.

`hipsparseLtComputetype_t`

The enumerator specifies the compute precision modes of the matrix

Value	Description
`HIPSPARSELT_COMPUTE_32F`	- Default mode for 32-bit floating-point precision - All computations and intermediate storage ensure at least 32-bit precision - Matrix Core will be used whenever possible (ROC only)
`HIPSPARSELT_COMPUTE_32I`	- Default mode for 32-bit integer precision - All computations and intermediate storage ensure at least 32-bit integer precision - Matrix Core / Tensor Core will be used whenever possible
`HIPSPARSELT_COMPUTE_16F`	- Default mode for 16-bit floating-point precision - All computations and intermediate storage ensure at least 16-bit precision - Matrix Core / Tensor Core will be used whenever possible
`HIPSPARSELT_COMPUTE_TF32_FAST`	- Default mode for 32-bit floating-point precision - The inputs are supposed to be directly represented in TensorFloat-32 precision. The 32-bit floating-point values are truncated to TensorFloat-32 before the computation

Value	Description
	- All computations and intermediate storage ensure at least TensorFloat-32 precision - Tensor Cores will be used whenever possible (CUDA only)
`HIPSPARSELT_COMPUTE_TF32`	- All computations and intermediate storage ensure at least TensorFloat-32 precision - The inputs are rounded to TensorFloat-32 precision. This mode is slower than `HIPSPARSELT_COMPUTE_TF32_FAST`, but could provide more accurate results - Tensor Cores will be used whenever possible (CUDA only)

The compute precision is used in the hipsparseLtMatmulDescriptorInit() function.

`hipsparseLtMatDescAttribute_t`

The enumerator specifies the additional attributes of a matrix descriptor

Value	Description
`HIPSPARSELT_MAT_NUM_BATCHES`	Number of matrices in a batch (`int` data type)
`HIPSPARSELT_MAT_BATCH_STRIDE`	Stride between consecutive matrices in a batch expressed in terms of matrix elements (`int64_t` data type)

The algorithm enumerator is used in the hipsparseLtMatDescSetAttribute() and hipsparseLtMatDescGetAttribute() functions.

`hipsparseLtMatmulDescAttribute_t`

The enumerator specifies the additional attributes of a matrix multiplication descriptor

Value	Type	Default Value	Description
`HIPSPARSELT_MATMUL_ACTIVATION_RELU`	`int` 0: **false**, **true** otherwise	`false`	ReLU activation function
`HIPSPARSELT_MATMUL_ACTIVATION_RELU_UPPERBOUND`	`float`	`inf`	Upper bound of the ReLU activation function
`HIPSPARSELT_MATMUL_ACTIVATION_RELU_THRESHOLD`	`float`	`0.0f`	Lower threshold of the ReLU activation function
`HIPSPARSELT_MATMUL_ACTIVATION_GELU`	`int` 0: **false**, **true** otherwise	`false`	GeLU activation function
`HIPSPARSELT_MATMUL_ACTIVATION_ABS`	`int` 0: **false**, **true** otherwise	`false`	ABS activation function (ROC only)
`HIPSPARSELT_MATMUL_ACTIVATION_LEAKYRELU`	`int` 0: **false**, **true** otherwise	`false`	LeakyReLU activation function (ROC only)
`HIPSPARSELT_MATMUL_ACTIVATION_LEAKYRELU_ALPHA`	`float`	`1.0f`	Alpha value of the LeakyReLU activation function (ROC only)
`HIPSPARSELT_MATMUL_ACTIVATION_SIGMOID`	`int` 0: **false**, **true** otherwise	`false`	Sigmoid activation function (ROC only)
`HIPSPARSELT_MATMUL_ACTIVATION_TANH`	`int` 0: **false**, **true** otherwise	`false`	Tanh activation function (ROC only)
`HIPSPARSELT_MATMUL_ACTIVATION_TANH_ALPHA`	`float`	`1.0f`	Alpha value of the Tanh activation function (ROC only)
`HIPSPARSELT_MATMUL_ACTIVATION_TANH_BETA`	`float`	`1.0f`	Beta value of the Tanh activation function (ROC only)

where the $ReLU$ activation function is defined as:

$$\text{ReLU(v)} = \begin{cases} v > threshold, & min(v, upperbound) \\ v \leq threshold, & 0 \end{cases}$$

The algorithm enumerator is used in the hipsparseLtMatmulDescSetAttribute() and hipsparseLtMatmulDescGetAttribute() functions.

`hipsparseLtMatmulAlg_t`

The enumerator specifies the algorithm for matrix-matrix multiplication

Value	Description
`HIPSPARSELT_MATMUL_ALG_DEFAULT`	Default algorithm

The algorithm enumerator is used in the hipsparseLtMatmulAlgSelectionInit() function.

`hipsparseLtMatmulAlgAttribute_t`

The enumerator specifies the matrix multiplication algorithm attributes

Value	Description
`HIPSPARSELT_MATMUL_ALG_CONFIG_ID`	Algorithm ID (set and query)
`HIPSPARSELT_MATMUL_ALG_CONFIG_MAX_ID`	Algorithm ID limit (query only)
`HIPSPARSELT_MATMUL_SEARCH_ITERATIONS`	Number of iterations (kernel launches per algorithm) for hipsparseLtMatmulSearch(), default=10

The algorithm attribute enumerator is used in the hipsparseLtMatmulAlgGetAttribute() and hipsparseLtMatmulAlgSetAttribute() functions.

`hipsparseLtPruneAlg_t`

The enumerator specifies the pruning algorithm to apply to the structured matrix before the compression

Value	Description
`HIPSPARSELT_PRUNE_SPMMA_TILE`	- `half`, `bfloat16`, `int8`: Zero-out eight values in a 4x4 tile to maximize the *L1-norm* of the resulting tile, under the constraint of selecting exactly two elements for each row and column - `float`, `tf32`: Zero-out two values in a 2x2 tile to maximize the *L1-norm* of the resulting tile, under the constraint of selecting exactly one element for each row and column (CUDA only)
`HIPSPARSELT_PRUNE_SPMMA_STRIP`	- `half`, `bfloat16`, `int8` - `float8`, `bfloat8` (ROC only) Zero-out two values in a 1x4 strip to maximize the L1-norm of the resulting strip The strip direction is chosen according to the operation `op` and matrix layout applied to the structured (sparse) matrix - `float`, `tf32`: Zero-out one value in a 1x2 strip to maximize the *L1-norm* of the resulting strip The strip direction is chosen according to the operation `op` and matrix layout applied to the structured (sparse) matrix (CUDA only)

The pruning algorithm is used in the hipsparseLtSpMMAPrune() function.

hipSPARSELt Functions

Library Management Functions

`hipsparseLtInit`

```
hipsparseLtStatus_t
hipsparseLtInit(hipsparseLtHandle_t* handle)
```

The function initializes the hipsparselt library handle (`hipsparseLtHandle_t`) which holds the hipsparselt library context. It allocates light hardware resources on the host, and must be called prior to making any other hipsparselt library calls. Calling any hipsparselt function which uses `hipsparseLtHandle_t` without a previous call of `hipsparseLtInit()` will return an error.

The hipsparselt library context is tied to the current ROCm/CUDA device. To use the library on multiple devices, one hipsparselt handle should be created for each device.

Parameter	Memory	In/Out	Description
`handle`	Host	OUT	hipsparselt library handle

See hipsparseLtStatus_t for the description of the return status.

`hipsparseLtDestroy`

```
hipsparseLtStatus_t
hipsparseLtDestroy(const hipsparseLtHandle_t* handle)
```

The function releases hardware resources used by the hipsparselt library. This function is the last call with a particular handle to the hipsparselt library.

Calling any hipsparselt function which uses `hipsparseLtHandle_t` after `hipsparseLtDestroy()` will return an error.

Parameter	Memory	In/Out	Description
`handle`	Host	IN	hipsparselt library handle

See hipsparseLtStatus_t for the description of the return status.

Matrix Descriptor Functions

`hipsparseLtDenseDescriptorInit`

```
hipsparseLtStatus_t
hipsparseLtDenseDescriptorInit(const hipsparseLtHandle_t*  handle,
                               hipsparseLtMatDescriptor_t* matDescr,
                               int64_t                     rows,
                               int64_t                     cols,
                               int64_t                     ld,
                               uint32_t                    alignment,
                               hipsparseLtDatatype_t       valueType,
                               hipsparseLtOrder_t          order)
```

The function initializes the descriptor of a *dense* matrix.

Parameter	Memory	In/Out	Description	Possible Values
`handle`	Host	IN	hipsparselt library handle	
`matDescr`	Host	OUT	Dense matrix description	
`rows`	Host	IN	Number of rows	
`cols`	Host	IN	Number of columns	
`ld`	Host	IN	Leading dimension	≥ rows if column-major, ≥ cols if row-major
`alignment`	Host	IN	Memory alignment in bytes	Multiple of 16 (CUDA only)
`valueType`	Host	IN	Data type of the matrix	`HIPSPARSELT_R_32F` (CUDA only), `HIPSPARSELT_R_16F` , `HIPSPARSELT_R_16BF` , `HIPSPARSELT_R_8I` , `HIPSPARSELT_R_8F` (ROC only), `HIPSPARSELT_R_8BF` (ROC only)
`order`	Host	IN	Memory layout	`HIPSPARSELT_ORDER_COLUMN` , `HIPSPARSELT_ORDER_ROW` (CUDA only)

Constrains:

- **ROC Backend:**
  - `row`, `col` must ≥ 8
  - For matrix B = K x N, `K` must be a multiple of 8
- **CUDA Backend:**
  - `rows`, `cols`, and `ld` must be a multiple of
    - 16 if `valueType` is `HIPSPARSELT_R_8I`
    - 8 if `valueType` is `HIPSPARSELT_R_16F` or `HIPSPARSELT_R_16BF`
    - 4 if `valueType` is `HIPSPARSELT_R_32F`
  - The total size of the matrix cannot exceed:
    - 2^{32} - 1elements for `HIPSPARSELT_R_8I`
    - 2^{31} - 1 elements for `HIPSPARSELT_R_16F` or `HIPSPARSELT_R_16BF`
    - 2^{30} - 1 elements for `HIPSPARSELT_R_32F`

See hipsparseLtStatus_t for the description of the return status.

`hipsparseLtStructuredDescriptorInit`

```
hipsparseLtStatus_t
hipsparseLtStructuredDescriptorInit(const hipsparseLtHandle_t*  handle,
                                    hipsparseLtMatDescriptor_t* matDescr,
                                    int64_t                     rows,
                                    int64_t                     cols,
                                    int64_t                     ld,
                                    uint32_t                    alignment,
                                    hipsparseLtDatatype_t       valueType,
                                    hipsparseLtOrder_t          order,
                                    hipsparseLtSparsity_t       sparsity)
```

The function initializes the descriptor of a *structured* matrix.

Parameter	Memory	In/Out	Description	Possible Values
`handle`	Host	IN	hipsparselt library handle	
`matDescr`	Host	OUT	Dense matrix description	

Parameter	Memory	In/Out	Description	Possible Values
`rows`	Host	IN	Number of rows	
`cols`	Host	IN	Number of columns	
`ld`	Host	IN	Leading dimension	≥ rows if column-major, ≥ cols if row-major
`alignment`	Host	IN	Memory alignment in bytes	Multiple of 16 (CUDA only)
`valueType`	Host	IN	Data type of the matrix	`HIPSPARSELT_R_32F` (CUDA only), `HIPSPARSELT_R_16F` , `HIPSPARSELT_R_16BF` , `HIPSPARSELT_R_8I` , `HIPSPARSELT_R_8F` (ROC only), `HIPSPARSELT_R_8BF` (ROC only)
`order`	Host	IN	Memory layout	`HIPSPARSELT_ORDER_COLUMN` , `HIPSPARSELT_ORDER_ROW` (CUDA only)
`sparsity`	Host	IN	Matrix sparsity ratio	`HIPSPARSELT_SPARSITY_50_PERCENT`

Constrains:

- ROC Backend:
  - `row`, `col` must ≥ 8
  - For `op = HIPSPARSELT_OPERATION_NON_TRANSPOSE`
    - `col` must be the multiplication of 8
  - For `op = HIPSPARSELT_OPERATION_TRANSPOSE`
    - `row` must be the multiplication of 8
- CUDA Backend:
  - `rows`, `cols`, and `ld` must be a multiple of
    - 32 if `valueType` is `HIPSPARSELT_R_8I`

    - 8 if `valueType` is `HIPSPARSELT_R_16F` or `HIPSPARSELT_R_16BF`
    - 4 if `valueType` is `HIPSPARSELT_R_32F`
  - The total size of the matrix cannot exceed:
    - 2^{32} - 1elements for `HIPSPARSELT_R_8I`
    - 2^{31} - 1 elements for `HIPSPARSELT_R_16F` or `HIPSPARSELT_R_16BF`
    - 2^{30} - 1 elements for `HIPSPARSELT_R_32F`
-

See hipsparseLtStatus_t for the description of the return status.

`hipsparseLtMatDescriptorDestroy`

```
hipsparseLtStatus_t
hipsparseLtMatDescriptorDestroy(const hipsparseLtMatDescriptor_t* matDescr)
```

The function releases the resources used by an instance of a matrix descriptor. After this call, the matrix descriptor and the matmul descriptor can no longer be used.

Parameter	Memory	In/Out	Description
`matDescr`	Host	IN	Matrix descriptor

See hipsparseLtStatus_t for the description of the return status.

`hipsparseLtMatDescSetAttribute`

```
hipsparseLtStatus_t
hipsparseLtMatDescSetAttribute(const hipsparseLtHandle_t*     handle,
                               hipsparseLtMatDescriptor_t*    matmulDescr,
                               hipsparseLtMatDescAttribute_t  matAttribute,
                               const void*                    data,
                               size_t                         dataSize)
```

The function sets the value of the specified attribute belonging to matrix descriptor such as number of batches and their stride.

Parameter	Memory	In/Out	Description	Possible Values
`handle`	Host	IN	hipsparselt library handle	
`matmulDescr`	Host	OUT	Matrix descriptor	
`matAttribute`	Host	IN	Attribute to set	`HIPSPARSELT_MAT_NUM_BATCHES` , `HIPSPARSELT_MAT_BATCH_STRIDE`
`data`	Host	IN	Pointer to the value to which the specified attribute will be set	
`dataSize`	Host	IN	Size in bytes of the attribute value used for verification	

See hipsparseLtStatus_t for the description of the return status.

`hipsparseLtMatDescGetAttribute`

```
hipsparseLtStatus_t
hipsparseLtMatDescGetAttribute(const hipsparseLtHandle_t*        handle,
                               const hipsparseLtMatDescriptor_t* matmulDes
                               hipsparseLtMatDescAttribute_t     matAttrib
                               void*                             data,
                               size_t                            dataSize)
```

The function gets the value of the specified attribute belonging to matrix descriptor such as number of batches and their stride.

Parameter	Memory	In/Out	Description	Possible Values
`handle`	Host	IN	hipsparselt library handle	
`matmulDescr`	Host	IN	Matrix descriptor	
`matAttribute`	Host	IN	Attribute to retrieve	`HIPSPARSELT_MAT_NUM_BATCHES`, `HIPSPARSELT_MAT_BATCH_STRIDE`
`data`	Host	OUT	Memory address containing the attribute value retrieved by this function	
`dataSize`	Host	IN	Size in bytes of the attribute value used for verification	

See hipsparseLtStatus_t for the description of the return status.

Matmul Descriptor Functions

hipsparseLtMatmulDescriptorInit

```
hipsparseLtStatus_t
hipsparseLtMatmulDescriptorInit(const hipsparseLtHandle_t*        handle,
                                hipsparseLtMatmulDescriptor_t*    matmulDescr,
                                hipsparseLtOperation_t            opA,
                                hipsparseLtOperation_t            opB,
                                const hipsparseLtMatDescriptor_t* matA,
                                const hipsparseLtMatDescriptor_t* matB,
                                const hipsparseLtMatDescriptor_t* matC,
                                const hipsparseLtMatDescriptor_t* matD,
                                hipsparseLtComputetype_t          computeType)
```

The function initializes the *matrix multiplication* descriptor.

Parameter	Memory	In/Out	Description	Possible Values
`handle`	Host	IN	hipsparselt library handle	
`matmulDescr`	Host	OUT	Matrix multiplication descriptor	
`opA`	Host	IN	Operation applied to the matrix `A`	`HIPSPARSELT_OPERATION_NON_TRANSPOSE`, `HIPSPARSELT_OPERATION_TRANSPOSE`
`opB`	Host	IN	Operation applied to the matrix `B`	`HIPSPARSELT_OPERATION_NON_TRANSPOSE`, `HIPSPARSELT_OPERATION_TRANSPOSE`
`matA`	Host	IN	Structured matrix descriptor `A`	
`matB`	Host	IN	Dense matrix descriptor `B`	
`matC`	Host	IN	Dense matrix descriptor `C`	
`matD`	Host	IN	Dense matrix descriptor `D`	
`computeType`	Host	IN	Compute precision	`HIPSPARSELT_COMPUTE_32F8F` (ROC only), `HIPSPARSELT_COMPUTE_32I`, `HIPSPARSELT_COMPUTE_16F` (CUDA only),

Parameter	Memory	In/Out	Description	Possible Values
				`HIPSPARSELT_COMPUTE_TF32` (CUDA only), `HIPSPARSELT_COMPUTE_TF32_FAST` (CUDA only)

The structured matrix descriptor can used for `matA` or `matB` but not both.

Data types Supported:

- ROC Backend:

Input	Output	Compute
`HIPSPARSELT_R_16F`	`HIPSPARSELT_R_16F`	`HIPSPARSELT_COMPUTE_32F`
`HIPSPARSELT_R_16BF`	`HIPSPARSELT_R_16BF`	`HIPSPARSELT_COMPUTE_32F`
`HIPSPARSELT_R_8I`	`HIPSPARSELT_R_8I`	`HIPSPARSELT_COMPUTE_32I`
`HIPSPARSELT_R_8F`	`HIPSPARSELT_R_8F`	`HIPSPARSELT_COMPUTE_32F`
`HIPSPARSELT_R_8BF`	`HIPSPARSELT_R_8BF`	`HIPSPARSELT_COMPUTE_32F`

- CUDA Backend:

Input	Output	Compute
`HIPSPARSELT_R_16F`	`HIPSPARSELT_R_16F`	`HIPSPARSELT_COMPUTE_16F`
`HIPSPARSELT_R_16BF`	`HIPSPARSELT_R_16BF`	`HIPSPARSELT_COMPUTE_16F`
`HIPSPARSELT_R_8I`	`HIPSPARSELT_R_8I`	`HIPSPARSELT_COMPUTE_32I`
`HIPSPARSELT_R_32F`	`HIPSPARSELT_R_32F`	`HIPSPARSELT_COMPUTE_TF32_FAST`
`HIPSPARSELT_R_32F`	`HIPSPARSELT_R_32F`	`HIPSPARSELT_COMPUTE_TF32`

See hipsparseLtStatus_t for the description of the return status.

`hipsparseLtMatmulDescSetAttribute`

```
hipsparseLtStatus_t
hipsparseLtMatmulDescSetAttribute(const hipsparseLtHandle_t*       handle,
                                  hipsparseLtMatmulDescriptor_t*   matmulDescr,
                                  hipsparseLtMatmulDescAttribute_t matmulAttribute,
                                  const void*                      data,
                                  size_t                           dataSize)
```

The function sets the value of the specified attribute belonging to matrix descriptor such as activation function and bias.

Parameter	Memory	In/ Out	Description	
`handle`	Host	IN	hipsparselt library handle	
`matmulDescr`	Host	OUT	Matrix descriptor	

`matmulAttribute`	Host	IN	Attribute to set	`HIPSPARSELT_MATMUL_ACTIVATION_RELU, HIPSPARSELT_MATMUL_ACTIVATION_RELU_UPPERBOUND, HIPSPARSELT_MATMUL_ACTIVATION_RELU_THRESHOLD, HIPSPARSELT_MATMUL_ACTIVATION_GELU, HIPSPARSELT_MATMUL_BIAS_POINTER, HIPSPARSELT_MATMUL_BIAS_STRIDE`
				ROC Only:
				`HIPSPARSELT_MATMUL_ACTIVATION_ABS,`
				`HIPSPARSELT_MATMUL_ACTIVATION_LEAKYRELU,`
				`HIPSPARSELT_MATMUL_ACTIVATION_LEAKYRELU_ALPHA,`
				`HIPSPARSELT_MATMUL_ACTIVATION_SIGMOID,`
				`HIPSPARSELT_MATMUL_ACTIVATION_TANH,`
				`HIPSPARSELT_MATMUL_ACTIVATION_TANH_ALPHA,`
				`HIPSPARSELT_MATMUL_ACTIVATION_TANH_BETA`

Parameter	Memory	In/ Out	Description	
`data`	Host	IN	Pointer to the value to which the specified attribute will be set	
`dataSize`	Host	IN	Size in bytes of the attribute value used for verification	

See hipsparseLtStatus_t for the description of the return status.

`hipsparseLtMatmulDescGetAttribute`

```
hipsparseLtStatus_t
hipsparseLtMatmulDescGetAttribute(const hipsparseLtHandle_t*           handle,
                                  const hipsparseLtMatmulDescriptor_t* matmulDescr,
                                  hipsparseLtMatmulDescAttribute_t     matmulAttribute,
                                  void*                                data,
                                  size_t                               dataSize)
```

The function gets the value of the specified attribute belonging to matrix descriptor such as activation function and bias.

Parameter	Memory	In/Out	Description	
`handle`	Host	IN	hipsparselt library handle	
`matmulDescr`	Host	IN	Matrix descriptor	

`matmulAttribute`	Host	IN	Attribute to retrieve	`HIPSPARSELT_MATMUL_ACTIVATION_RELU,`
				`HIPSPARSELT_MATMUL_ACTIVATION_RELU_UPPERBOUND,`
				`HIPSPARSELT_MATMUL_ACTIVATION_RELU_THRESHOLD, HIPSPARSELT_MATMUL_ACTIVATION_GELU,`

Parameter	Memory	In/Out	Description	

				`HIPSPARSELT_MATMUL_BIAS_POINTER, HIPSPARSELT_MATMUL_BIAS_STRIDE`
				ROC Only:
				`HIPSPARSELT_MATMUL_ACTIVATION_ABS, HIPSPARSELT_MATMUL_ACTIVATION_LEAKYRELU, HIPSPARSELT_MATMUL_ACTIVATION_LEAKYRELU_ALPHA, HIPSPARSELT_MATMUL_ACTIVATION_SIGMOID, HIPSPARSELT_MATMUL_ACTIVATION_TANH, HIPSPARSELT_MATMUL_ACTIVATION_TANH_ALPHA, HIPSPARSELT_MATMUL_ACTIVATION_TANH_BETA`

`data`	Host	OUT	Memory address containing the attribute value retrieved by this function	
`dataSize`	Host	IN	Size in bytes of the attribute value used for verification	

See hipsparseLtStatus_t for the description of the return status.

Matmul Algorithm Functions

hipsparseLtMatmulAlgSelectionInit

```
hipsparseLtStatus_t
hipsparseLtMatmulAlgSelectionInit(const hipsparseLtHandle_t*           handle,
                                  hipsparseLtMatmulAlgSelection_t*     algSelection,
                                  const hipsparseLtMatmulDescriptor_t* matmulDescr,
                                  hipsparseLtMatmulAlg_t               alg)
```

The function initializes the *algorithm selection* descriptor.

Parameter	Memory	In/Out	Description	Possible Values
`handle`	Host	IN	hipsparselt library handle	
`algSelection`	Host	OUT	Algorithm selection descriptor	
`matmulDescr`	Host	IN	Matrix multiplication descriptor	
`alg`	Host	IN	Algorithm mode	`HIPSPARSELT_MATMUL_ALG_DEFAULT`

See hipsparseLtStatus_t for the description of the return status.

`hipsparseLtMatmulAlgSetAttribute`

```
hipsparseLtStatus_t
hipsparseLtMatmulAlgSetAttribute(const hipsparseLtHandle_t*       handle,
                                 hipsparseLtMatmulAlgSelection_t* algSelect
                                 hipsparseLtMatmulAlgAttribute_t  attribute
                                 const void*                      data,
                                 size_t                           dataSize)
```

The function sets the value of the specified attribute belonging to algorithm selection descriptor.

Parameter	Memory	In/Out	Description	Possible Values
`handle`	Host	IN	hipsparselt library handle	
`algSelection`	Host	OUT	Algorithm selection descriptor	
`attribute`	Host	IN	The attribute to set	`HIPSPARSELT_MATMUL_ALG_CONFIG_ID`, `HIPSPARSELT_MATMUL_ALG_CONFIG_MAX_ID`, `HIPSPARSELT_MATMUL_SEARCH_ITERATIONS`
`data`	Host	IN	Pointer to the value to which the specified attribute will be set	
`dataSize`	Host	IN	Size in bytes of the attribute value used for verification	

See hipsparseLtStatus_t for the description of the return status.

hipsparseLtMatmulAlgGetAttribute

```
hipsparseLtStatus_t
hipsparseLtMatmulAlgGetAttribute(const hipsparseLtHandle_t*             handle,
                                 const hipsparseLtMatmulAlgSelection_t* algSelection,
                                 hipsparseLtMatmulAlgAttribute_t        attribute,
                                 void*                                  data,
                                 size_t                                 dataSize)
```

The function returns the value of the queried attribute belonging to algorithm selection descriptor.

Parameter	Memory	In/Out	Description	Possible Values
`handle`	Host	IN	hipsparselt library handle	
`algSelection`	Host	IN	Algorithm selection descriptor	
`attribute`	Host	IN	The attribute that will be retrieved by this function	`HIPSPARSELT_MATMUL_ALG_CONFIG_ID`, `HIPSPARSELT_MATMUL_ALG_CONFIG_MAX_ID`, `HIPSPARSELT_MATMUL_SEARCH_ITERATIONS`
`data`	Host	OUT	Memory address containing the attribute value retrieved by this function	
`dataSize`	Host	IN	Size in bytes of the attribute value used for verification	

See hipsparseLtStatus_t for the description of the return status.

Matmul Functions

`hipsparseLtMatmulGetWorkspace`

```
hipsparseLtStatus_t
hipsparseLtMatmulGetWorkspace(const hipsparseLtHandle_t*      handle,
                              const hipsparseLtMatmulPlan _t* plan,
                              size_t*                         workspaceSize)
```

The function determines the required workspace size associated to the selected algorithm.

Parameter	Memory	In/Out	Description
`handle`	Host	IN	hipsparselt library handle
`plan`	Host	IN	Matrix multiplication plan
`workspaceSize`	Host	OUT	Workspace size in bytes

See hipsparseLtStatus_t for the description of the return status.

`hipsparseLtMatmulPlanInit`

```
hipsparseLtStatus_t
hipsparseLtMatmulPlanInit(const hipsparseLtHandle_t*             handle,
                          hipsparseLtMatmulPlan_t*               plan,
                          const hipsparseLtMatmulDescriptor_t*   matmulDescr,
                          const hipsparseLtMatmulAlgSelection_t* algSelection,
                          size_t                                 workspaceSize)
```

Parameter	Memory	In/Out	Description
`handle`	Host	IN	hipsparselt library handle
`plan`	Host	OUT	Matrix multiplication plan
`matmulDescr`	Host	IN	Matrix multiplication descriptor
`algSelection`	Host	IN	Algorithm selection descriptor
`workspaceSize`	Host	IN	Workspace size in bytes

See hipsparseLtStatus_t for the description of the return status.

`hipsparseLtMatmulPlanDestroy`

```
hipsparseLtStatus_t
hipsparseLtMatmulPlanDestroy(const hipsparseLtMatmulPlan_t* plan)
```

The function releases the resources used by an instance of the matrix multiplication plan. This function is the last call with a specific plan instance.

Calling any hipsparselt function which uses `hipsparseLtMatmulPlan_t` after `hipsparseLtMatmulPlanDestroy()` will return an error.

Parameter	Memory	In/Out	Description
`plan`	Host	IN	Matrix multiplication plan

See hipsparseLtStatus_t for the description of the return status.

`hipsparseLtMatmul`

```
hipsparseLtStatus_t
hipsparseLtMatmul(const hipsparseLtHandle_t*     handle,
                  const hipsparseLtMatmulPlan_t* plan,
                  const void*                    alpha,
                  const void*                    d_A,
                  const void*                    d_B,
                  const void*                    beta,
                  const void*                    d_C,
                  void*                          d_D,
                  void*                          workspace,
                  hipStream_t*                   streams,
                  int32_t                        numStreams)
```

The function computes the matrix multiplication of matrices `A` and `B` to produce the output matrix `D`, according to the following operation:

$$D = \mathrm{Activation}(\ \alpha op(A) * op(B) + \beta C + bias)$$

where `A`, `B`, and `C` are input matrices, and α and β are input scalars.

Note: The function currently only supports the case where `D` has the same shape of `C`

Parameter	Memory	In/Out	Description
`handle`	Host	IN	hipsparselt library handle
`plan`	Host	IN	Matrix multiplication plan
`alpha`	Host	IN	α scalar used for multiplication (`float` data type)
`d_A`	Device	IN	Pointer to the structured matrix `A`
`d_B`	Device	IN	Pointer to the dense matrix `B`
`beta`	Host	IN	β scalar used for multiplication (`float` data type)
`d_C`	Device	OUT	Pointer to the dense matrix `C`
`d_D`	Device	OUT	Pointer to the dense matrix `D`
`workspace`	Device	IN	Pointer to workspace
`streams`	Host	IN	Pointer to HIP stream array for the computation
`numStreams`	Host	IN	Number of HIP streams in `streams`

Data types Supported:

- ROC Backend:

Input	Output	Compute
`HIPSPARSELT_R_16F`	`HIPSPARSELT_R_16F`	`HIPSPARSELT_COMPUTE_32F`
`HIPSPARSELT_R_16BF`	`HIPSPARSELT_R_16BF`	`HIPSPARSELT_COMPUTE_32F`
`HIPSPARSELT_R_8I`	`HIPSPARSELT_R_8I`	`HIPSPARSELT_COMPUTE_32I`
`HIPSPARSELT_R_8F`	`HIPSPARSELT_R_8F`	`HIPSPARSELT_COMPUTE_32F`
`HIPSPARSELT_R_8BF`	`HIPSPARSELT_R_8BF`	`HIPSPARSELT_COMPUTE_32F`

- CUDA Backend:

Input	Output	Compute
`HIPSPARSELT_R_16F`	`HIPSPARSELT_R_16F`	`HIPSPARSELT_COMPUTE_16F`
`HIPSPARSELT_R_16BF`	`HIPSPARSELT_R_16BF`	`HIPSPARSELT_COMPUTE_16F`
`HIPSPARSELT_R_8I`	`HIPSPARSELT_R_8I`	`HIPSPARSELT_COMPUTE_32I`
`HIPSPARSELT_R_32F`	`HIPSPARSELT_R_32F`	`HIPSPARSELT_COMPUTE_TF32_FAST`
`HIPSPARSELT_R_32F`	`HIPSPARSELT_R_32F`	`HIPSPARSELT_COMPUTE_TF32`

The *structured matrix* `A` (before the compression) must respect the following constrains depending on the operation applied on it:

- For `op = HIPSPARSELT_OPERATION_NON_TRANSPOSE`
  - `HIPSPARSELT_R_16F`, `HIPSPARSELT_R_16BF`, `HIPSPARSELT_R_8I`, `HIPSPARSELT_R_8F`, `HIPSPARSELT_R_8BF` each row must have at least two zero values every four elements
  - `HIPSPARSELT_R_32F` each row must have at least one zero values every two elements
- For `op = HIPSPARSELT_OPERATION_TRANSPOSE`
  - `HIPSPARSELT_R_16F`, `HIPSPARSELT_R_16BF`, `HIPSPARSELT_R_8I`, `HIPSPARSELT_R_8F`, `HIPSPARSELT_R_8BF` each column must have at least two zero values every four elements
  - `HIPSPARSELT_R_32F` each column must have at least one zero values every two elements

The correctness of the pruning result (matrix `A`) can be check with the function hipsparseLtSpMMAPruneCheck().

Properties

- The routine requires no extra storage
- The routine supports asynchronous execution with respect to `streams[0]`

See hipsparseLtStatus_t for the description of the return status.

`hipsparseLtMatmulSearch`

```
hipsparseLtStatus_t
hipsparseLtMatmulSearch(const hipsparseLtHandle_t* handle,
                        hipsparseLtMatmulPlan_t*   plan,
                        const void*                alpha,
                        const void*                d_A,
                        const void*                d_B,
                        const void*                beta,
                        const void*                d_C,
                        void*                      d_D,
                        void*                      workspace,
                        hipStream_t*               streams,
                        int32_t                    numStreams)
```

The function evaluates all available algorithms for the matrix multiplication and automatically updates the `plan` by selecting the fastest one. The functionality is intended to be used for auto-tuning purposes when the same operation is repeated multiple times over different inputs.

The function behavior is the same of hipsparseLtMatmul().

- The function is *NOT* asynchronous with respect to `streams[0]` (*blocking call*)
- The number of iterations for the evaluation can be set by using hipsparseLtMatmulAlgSetAttribute() with `HIPSPARSELT_MATMUL_SEARCH_ITERATIONS`.
- The selected algorithm id can be retrieved by using hipsparseLtMatmulAlgGetAttribute() with `HIPSPARSELT_MATMUL_ALG_CONFIG_ID`.

Helper Functions

`hipsparseLtSpMMAPrune`

```
hipsparseLtStatus_t
hipsparseLtSpMMAPrune(const hipsparseLtHandle_t*           handle,
                      const hipsparseLtMatmulDescriptor_t* matmulDe
                      const void*                          d_in,
                      void*                                d_out,
                      hipsparseLtPruneAlg_t                pruneAlg
                      hipStream_t                          stream)
```

The function prunes a dense matrix `d_in` according to the specified algorithm `pruneAlg`.

Parameter	Memory	In/Out	Description	Possible Values
`handle`	Host	IN	hipsparselt library handle	
`matmulDescr`	Host	IN	Matrix multiplication descriptor	
`d_in`	Device	IN	Pointer to the dense matrix	
`d_out`	Device	OUT	Pointer to the pruned matrix	
`pruneAlg`	Device	IN	Pruning algorithm	`HIPSPARSELT_PRUNE_SPMMA_TILE`, `HIPSPARSELT_PRUNE_SPMMA_STRIP`
`stream`	Host	IN	HIP stream for the computation	

Properties

- The routine requires no extra storage
- The routine supports asynchronous execution with respect to `stream`

See hipsparseLtStatus_t for the description of the return status.

`hipsparseLtSpMMAPrune2`

```
hipsparseLtStatus_t
hipsparseLtSpMMAPrune2(const hipsparseLtHandle_t*        handle,
                       const hipsparseLtMatDescriptor_t* sparseMat
                       int                               isSparseA
                       hipsparseLtOperation_t            op,
                       const void*                       d_in,
                       void*                             d_out,
                       hipsparseLtPruneAlg_t             pruneAlg,
                       hipStream_t                       stream)
```

The function prunes a dense matrix `d_in` according to the specified algorithm `pruneAlg`.

Parameter	Memory	In/ Out	Description	Possible Values
`handle`	Host	IN	hipsparselt library handle	
`sparseMatDescr`	Host	IN	structured(sparse) matrix descriptor	
`isSparse`	Host	IN	specify if the structured (sparse) matrix is in the first position (matA or matB) (only support matA)	
`op`	Host	IN	operation that will be applied to the structured (sparse) matrix in the multiplication	
`d_in`	Device	IN	Pointer to the dense matrix	
`d_out`	Device	OUT	Pointer to the pruned matrix	
`pruneAlg`	Device	IN	Pruning algorithm	`hipsparselt_prune_smfmac_tile`, `hipsparselt_prune_smfmac_strip`
`stream`	Host	IN	HIP stream for the computation	

Properties

- The routine requires no extra storage
- The routine supports asynchronous execution with respect to `stream`

See hipsparseLtStatus_t for the description of the return status.

`hipsparseLtSpMMAPruneCheck`

```
hipsparseLtStatus_t
hipsparseLtSpMMAPruneCheck(const hipsparseLtHandle_t*           handle,
                           const hipsparseLtMatmulDescriptor_t* matmulD
                           const void*                          d_in,
                           int*                                 valid,
                           hipStream_t                          stream)
```

The function checks the correctness of the pruning structure for a given matrix.

Parameter	Memory	In/Out	Description
`handle`	Host	IN	hipsparselt library handle
`matmulDescr`	Host	IN	Matrix multiplication descriptor
`d_in`	Device	IN	Pointer to the matrix to check
`d_valid`	Device	OUT	Validation results (`0` correct, `1` wrong)
`stream`	Host	IN	HIP stream for the computation

See hipsparseLtStatus_t for the description of the return status.

`hipsparseLtSpMMAPruneCheck2`

```
hipsparseLtStatus_t
hipsparseLtSpMMAPruneCheck2(const hipsparseLtHandle_t*        handle,
                            const hipsparseLtMatDescriptor_t* sparseMa
                            int                               isSparse
                            hipsparseLtOperation_t            op,
                            const void*                       d_in,
                            int*                              d_valid,
                            hipStream_t                       stream)
```

The function checks the correctness of the pruning structure for a given matrix.

Parameter	Memory	In/Out	Description
`handle`	Host	IN	hipsparselt library handle
`sparseMatDescr`	Host	IN	structured(sparse) matrix descriptor
`isSparse`	Host	IN	specify if the structured (sparse) matrix is in the first position (matA or matB) (only support matA)
`op`	Host	IN	operation that will be applied to the structured (sparse) matrix in the multiplication
`d_in`	Device	IN	Pointer to the matrix to check
`d_valid`	Device	OUT	Validation results (`0` correct, `1` wrong)
`stream`	Host	IN	HIP stream for the computation

See hipsparseLtStatus_t for the description of the return status.

`hipsparseLtSpMMACompressedSize`

```
hipsparseLtStatus_t
hipsparseLtSpMMACompressedSize(const hipsparseLtHandle_t*     handle,
                               const hipsparseLtMatmulPlan_t* plan,
                               size_t*                        compressedSize)
```

The function provides the size of the *compressed* matrix to be allocated before calling hipsparseLtSpMMACompress().

Parameter	Memory	In/Out	Description
`handle`	Host	IN	hipsparselt library handle
`plan`	Host	IN	Matrix plan descriptor
`compressedSize`	Host	OUT	Size in bytes of the compressed matrix

See hipsparseLtStatus_t for the description of the return status.

`hipsparseLtSpMMACompressedSize2`

```
hipsparseLtStatus_t
hipsparseLtSpMMACompressedSize2(const hipsparseLtHandle_t*        handle,
                                const hipsparseLtMatDescriptor_t* sparseMatDescr,
                                size_t*                           compressedSize)
```

The function provides the size of the *compressed* matrix to be allocated before calling hipsparselt_smfmac_compress().

Parameter	Memory	In/Out	Description
`handle`	Host	IN	hipsparselt library handle
`sparseMatDescr`	Host	IN	structured(sparse) matrix descriptor
`compressedSize`	Host	OUT	Size in bytes of the compressed matrix

See hipsparseLtStatus_t for the description of the return status.

`hipsparseLtSpMMACompress`

```
hipsparseLtStatus_t
hipsparseLtSpMMACompress(const hipsparseLtHandle_t*     handle,
                         const hipsparseLtMatmulPlan_t* plan,
                         const void*                    d_dense,
                         void*                          d_compressed,
hipsparseLtSpMMACompress(const hipsparseLtHandle_t*      handle,
                            const hipsparseLtMatmulPlan_t* plan,
                         hipStream_t                    stream)
```

The function compresses a dense matrix `d_dense`. The *compressed* matrix is intended to be used as the first operand `A` in the hipsparseLtMatmull() function.

Parameter	Memory	In/Out	Description
`handle`	Host	IN	hipsparselt library handle
`plan`	Host	IN	Matrix multiplication plan
`d_dense`	Device	IN	Pointer to the dense matrix
`d_compressed`	Device	OUT	Pointer to the *compressed* matrix

Parameter	Memory	In/Out	Description
`stream`	Host	IN	HIP stream for the computation

Properties

- The routine requires no extra storage
- The routine supports asynchronous execution with respect to `stream`

See hipsparseLtStatus_t for the description of the return status.

`hipsparseLtSpMMACompress2`

```
hipsparseLtStatus_t
hipsparseLtSpMMACompress2(const hipsparseLtHandle_t*        handle,
                          const hipsparseLtMatDescriptor_t* sparseMatDescr
                          int                               isSparseA,
                          hipsparseLtOperation_t            op,
                          const void*                       d_dense,
                          void*                             d_compressed,
                          hipStream_t                       stream)
```

The function compresses a dense matrix `d_dense`. The *compressed* matrix is intended to be used as the first operand `A` in the hipsparselt_matmul() function.

Parameter	Memory	In/Out	Description
`handle`	Host	IN	hipsparselt library handle
`sparseMatDescr`	Host	IN	structured(sparse) matrix descriptor
`isSparse`	Host	IN	specify if the structured (sparse) matrix is in the first position (matA or matB) (only support matA)
`op`	Host	IN	operation that will be applied to the structured (sparse) matrix in the multiplication
`d_dense`	Device	IN	Pointer to the dense matrix
`d_compressed`	Device	OUT	Pointer to the *compressed* matrix
`stream`	Host	IN	HIP stream for the computation

Properties

- The routine requires no extra storage
- The routine supports asynchronous execution with respect to `stream`

See hipsparseLtStatus_t for the description of the return status.