

---

# **SPLA Documentation**

***Release 0.1.0***

**ETH Zurich, Simon Frasch**

**Apr 22, 2023**



## C++ API REFERENCE:

<b>1</b>	<b>Types</b>	<b>3</b>
<b>2</b>	<b>Context</b>	<b>5</b>
<b>3</b>	<b>Matrix Distribution</b>	<b>9</b>
<b>4</b>	<b>GEMM</b>	<b>13</b>
<b>5</b>	<b>GEMM - SSB</b>	<b>15</b>
<b>6</b>	<b>GEMM - SSBTR</b>	<b>17</b>
<b>7</b>	<b>GEMM - SBS</b>	<b>19</b>
<b>8</b>	<b>Exceptions</b>	<b>21</b>
<b>9</b>	<b>Context</b>	<b>25</b>
<b>10</b>	<b>Matrix Distribution</b>	<b>31</b>
<b>11</b>	<b>GEMM</b>	<b>35</b>
<b>12</b>	<b>GEMM - SSB</b>	<b>37</b>
<b>13</b>	<b>GEMM - SSBTR</b>	<b>41</b>
<b>14</b>	<b>GEMM - SBS</b>	<b>43</b>
<b>15</b>	<b>Errors</b>	<b>45</b>
	<b>Index</b>	<b>47</b>



SPLA - Specialized Parallel Linear Algebra



## TYPES

### Enums

enum **SplaDistributionType**

*Values:*

enumerator **SPLA\_DIST\_BLACS\_BLOCK\_CYCLIC**

Blacs block cyclic distribution.

enumerator **SPLA\_DIST\_MIRROR**

Mirror distribution, where each rank holds a full matrix copy.

enum **SplaProcessingUnit**

*Values:*

enumerator **SPLA\_PU\_HOST**

Host / CPU.

enumerator **SPLA\_PU\_GPU**

GPU.

enum **SplaOperation**

*Values:*

enumerator **SPLA\_OP\_NONE**

None.

enumerator **SPLA\_OP\_TRANSPOSE**

Transpose.

enumerator **SPLA\_OP\_CONJ\_TRANSPOSE**

Conjugate transpose.

enum **SplaFillMode**

*Values:*

enumerator **SPLA\_FILL\_MODE\_FULL**

Full matrix.

enumerator **SPLA\_FILL\_MODE\_UPPER**

Upper triangular matrix.

enumerator **SPLA\_FILL\_MODE\_LOWER**

Lower triangular matrix.



## CONTEXT

class **Context**

*#include <context.hpp>* *Context*, which provides configuration settings and reusable resources.

**Public Functions**

explicit **Context**(*SplaProcessingUnit* pu)

Constructor of *Context* with default configuration for given processing unit.

**Parameters**

**pu** – [in] Processing unit to be used for computations.

**Context**(*Context*&&) = default

Default move constructor.

**Context**(const *Context*&) = delete

Disabled copy constructor.

*Context* &**operator**=(*Context*&&) = default

Default move assignment operator.

*Context* &**operator**=(const *Context*&) = delete

Disabled copy assignment operator.

*SplaProcessingUnit* **processing\_unit**() const

Access a *Context* parameter.

**Returns**

Processing unit used.

**SPLA\_DEPRECATED int num\_threads** () const

Access a *Context* parameter.

**Returns**

Maximum number of threads used for computations.

int **num\_tiles**() const

Access a *Context* parameter.

**Returns**

Number of tiles used to overlap computation and communication.

int **tile\_size\_host()** const

Access a *Context* parameter.

**Returns**

Size of tiles on host. Used for partitioning communication.

int **tile\_size\_gpu()** const

Access a *Context* parameter.

**Returns**

Target size of tiles on GPU.

int **op\_threshold\_gpu()** const

Access a *Context* parameter.

**Returns**

Operations threshold, below which computation may be done on Host, even if processing unit is set to GPU. For GEMM, the number of operations is estimated as  $2mnk$ .

int **gpu\_device\_id()** const

Access a *Context* parameter.

**Returns**

Id of GPU used for computations. This is set as fixed parameter by query of device id at context creation.

std::uint\_least64\_t **allocated\_memory\_host()** const

Access a *Context* parameter.

**Returns**

Total allocated memory on host in bytes used for internal buffers. Does not include allocations through standard C++ allocators. May change with use of context.

std::uint\_least64\_t **allocated\_memory\_pinned()** const

Access a *Context* parameter.

**Returns**

Total allocated pinned memory on host in bytes used for internal buffers. Does not include allocations through standard C++ allocators. May change with use of context.

std::uint\_least64\_t **allocated\_memory\_gpu()** const

Access a *Context* parameter.

**Returns**

Total allocated memory on gpu in bytes used for internal buffers. Does not include allocations by device libraries like cuBLAS / rocBLAS. May change with use of context.

**SPLA\_DEPRECATED void set\_num\_threads (int numThreads)**

Set the number of threads to be used.

**Parameters**

**numThreads** – [in] Number of threads.

void **set\_num\_tiles**(int numTilesPerThread)

Set the number of tiles.

**Parameters**

**numTilesPerThread** – [in] Number of tiles.

void **set\_tile\_size\_host**(int tileSizeHost)

Set the tile size used for computations on host and partitioning of communication.

**Parameters**

**tileSizeHost** – [in] Tile size.

void **set\_op\_threshold\_gpu**(int opThresholdGPU)

Set the operations threshold, below which computation may be done on Host, even if processing unit is set to GPU.

For GEMM, the number of operations is estimated as  $2mnk$ .

**Parameters**

**opThresholdGPU** – [in] Threshold in number of operations.

void **set\_tile\_size\_gpu**(int tileSizeGPU)

Set the tile size used for computations on GPU.

**Parameters**

**tileSizeGPU** – [in] Tile size on GPU.

void **set\_alloc\_host**(std::function<void\*(std::size\_t)> allocateFunc, std::function<void(void\*)> deallocateFunc)

Set the allocation and deallocation functions for host memory.

Internal default uses a memory pool for better performance.

**Parameters**

- **allocateFunc** – [in] Function allocating given size in bytes.
- **deallocateFunc** – [in] Function to deallocate memory allocated using allocateFunc.

void **set\_alloc\_pinned**(std::function<void\*(std::size\_t)> allocateFunc, std::function<void(void\*)> deallocateFunc)

Set the allocation and deallocation functions for pinned host memory.

Internal default uses a memory pool for better performance.

**Parameters**

- **allocateFunc** – [in] Function allocating given size in bytes.
- **deallocateFunc** – [in] Function to deallocate memory allocated using allocateFunc.

void **set\_alloc\_gpu**(std::function<void\*(std::size\_t)> allocateFunc, std::function<void(void\*)> deallocateFunc)

Set the allocation and deallocation functions for gpu memory.

Internal default uses a memory pool for better performance.

**Parameters**

- **allocateFunc** – [in] Function allocating given size in bytes.
- **deallocateFunc** – [in] Function to deallocate memory allocated using allocateFunc.



## MATRIX DISTRIBUTION

class **MatrixDistribution**

*#include <matrix\_distribution.hpp>*

### Public Functions

**MatrixDistribution**(*MatrixDistribution*&&) = default

Default move constructor.

**MatrixDistribution**(const *MatrixDistribution*&) = default

Default copy constructor.

*MatrixDistribution* &**operator=**(*MatrixDistribution*&&) = default

Default move assignment operator.

*MatrixDistribution* &**operator=**(const *MatrixDistribution*&) = default

Default copy assignment operator.

int **proc\_grid\_rows**() const

Access a distribution parameter.

#### Returns

Number of rows in process grid.

int **proc\_grid\_cols**() const

Access a distribution parameter.

#### Returns

Number of columns in process grid.

int **row\_block\_size**() const

Access a distribution parameter.

#### Returns

Row block size used for matrix partitioning.

int **col\_block\_size**() const

Access a distribution parameter.

#### Returns

Column block size used for matrix partitioning.

*SplaDistributionType* **type**() const

Access a distribution parameter.

**Returns**

Distribution type

MPI\_Comm **comm**()

Access a distribution parameter.

**Returns**

Communicator used internally. Order of ranks may differ from communicator provided for creation of distribution.

void **set\_row\_block\_size**(int rowBlockSize)

Set row block size used for matrix partitioning.

**Parameters**

**rowBlockSize** – [in] Row block size.

void **set\_col\_block\_size**(int colBlockSize)

Set column block size used for matrix partitioning.

**Parameters**

**colBlockSize** – [in] Column block size.

## Public Static Functions

static *MatrixDistribution* **create\_blacs\_block\_cyclic**(MPI\_Comm comm, char order, int procGridRows, int procGridCols, int rowBlockSize, int colBlockSize)

Create a blacs block cyclic matrix distribution with row major or column major ordering of MPI ranks.

**Parameters**

- **comm** – [in] MPI communicator to be used.
- **order** – [in] Either 'R' for row major ordering or 'C' for column major ordering.
- **procGridRows** – [in] Number of rows in process grid.
- **procGridCols** – [in] Number of columns in process grid.
- **rowBlockSize** – [in] Row block size for matrix partitioning.
- **colBlockSize** – [in] Column block size for matrix partitioning.

static *MatrixDistribution* **create\_blacs\_block\_cyclic\_from\_mapping**(MPI\_Comm comm, const int \*mapping, int procGridRows, int procGridCols, int rowBlockSize, int colBlockSize)

Create a blacs block cyclic matrix distribution with given process grid mapping.

**Parameters**

- **comm** – [in] MPI communicator to be used.
- **mapping** – [in] Pointer to array of size procGridRows \* procGridCols mapping MPI ranks onto a column major process grid.
- **procGridRows** – [in] Number of rows in process grid.
- **procGridCols** – [in] Number of columns in process grid.

- **rowBlockSize** – [in] Row block size for matrix partitioning.
- **colBlockSize** – [in] Coloumn block size for matrix partitioning.

static *MatrixDistribution* **create\_mirror**(MPI\_Comm comm)

Create a mirror distribution, where the full matrix is stored on each MPI rank.

**Parameters**

**comm** – [in] MPI communicator to be used.





## GEMM

General matrix multiplication functions for locally computing  $C \leftarrow \alpha OP(A)OP(B) + \beta C$ .

### Functions

void **gemm**(*SplaOperation* opA, *SplaOperation* opB, int m, int n, int k, float alpha, const float \*A, int lda, const float \*B, int ldb, float beta, float \*C, int ldc, *Context* &ctx)

Computes a local general matrix multiplication of the form  $C \leftarrow \alpha op(A)op(B) + \beta C$  in single precision.

If context with processing unit set to GPU, pointers to matrices can be any combination of host and device pointers.

#### Parameters

- **opA** – [in] Operation applied when reading matrix  $A$ .
- **opB** – [in] Operation applied when reading matrix  $B$ .
- **m** – [in] Number of rows of  $OP(A)$
- **n** – [in] Number of columns of  $OP(B)$
- **k** – [in] Number rows of  $OP(B)$  and number of columns of  $OP(A)$
- **alpha** – [in] Scaling of multiplication of  $A^H$  and  $B$
- **A** – [in] Pointer to matrix  $A$ .
- **lda** – [in] Leading dimension of  $A$ .
- **B** – [in] Pointer to matrix  $B$ .
- **ldb** – [in] Leading dimension of  $B$ .
- **beta** – [in] Scaling of  $C$  before summation.
- **C** – [out] Pointer to matrix  $C$ .
- **ldc** – [in] Leading dimension of  $C$ .
- **ctx** – [in] *Context*, which provides configuration settings and reusable resources.

void **gemm**(*SplaOperation* opA, *SplaOperation* opB, int m, int n, int k, double alpha, const double \*A, int lda, const double \*B, int ldb, double beta, double \*C, int ldc, *Context* &ctx)

Computes a local general matrix multiplication of the form  $C \leftarrow \alpha OP(A)OP(B) + \beta C$  in double precision.

See documentation above.

```
void gemm(SplaOperation opA, SplaOperation opB, int m, int n, int k, std::complex<float> alpha, const  
         std::complex<float> *A, int lda, const std::complex<float> *B, int ldb, std::complex<float> beta,  
         std::complex<float> *C, int ldc, Context &ctx)
```

Computes a local general matrix multiplication of the form  $C \leftarrow \alpha OP(A)OP(B) + \beta C$  in single precision complex types.

See documentation above.

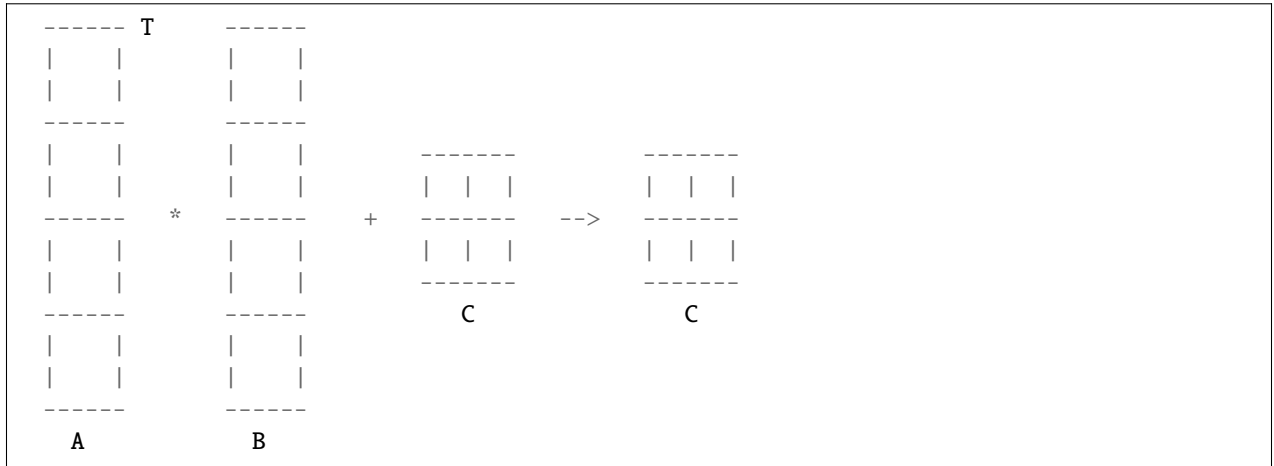
```
void gemm(SplaOperation opA, SplaOperation opB, int m, int n, int k, std::complex<double> alpha, const  
         std::complex<double> *A, int lda, const std::complex<double> *B, int ldb, std::complex<double> beta,  
         std::complex<double> *C, int ldc, Context &ctx)
```

Computes a local general matrix multiplication of the form  $C \leftarrow \alpha OP(A)OP(B) + \beta C$  in double precision complex types.

See documentation above.

## GEMM - SSB

General matrix multiplication functions for computing  $C \leftarrow \alpha A^H B + \beta C$  with stripe-stripe-block distribution.



### Functions

void **pgemm\_ssb**(int m, int n, int kLocal, *SplaOperation* opA, float alpha, const float \*A, int lda, const float \*B, int ldb, float beta, float \*C, int ldc, int cRowOffset, int cColOffset, *MatrixDistribution* &distC, *Context* &ctx)

Computes a distributed general matrix multiplication of the form  $C \leftarrow \alpha A^H B + \beta C$  in single precision.

$A$  and  $B$  are only split along the row dimension (stripes), while  $C$  can be distributed as any supported *MatrixDistribution* type.

#### Parameters

- **m** – [in] Number of rows of  $A^H$
- **n** – [in] Number of columns of  $B$
- **kLocal** – [in] Number rows of  $B$  and number of columns of  $A^H$  stored at calling MPI rank. This number may differ for each rank.
- **opA** – [in] Operation applied when reading matrix  $A$ . Must be SPLA\_OP\_TRANSPOSE or SPLA\_OP\_CONJ\_TRANSPOSE.
- **alpha** – [in] Scaling of multiplication of  $A^H$  and  $B$
- **A** – [in] Pointer to matrix  $A$ .
- **lda** – [in] Leading dimension of  $A$  with  $\text{lda} \geq \text{kLocal}$ .

- **B** – [in] Pointer to matrix  $B$ .
- **ldb** – [in] Leading dimension of  $B$  with  $\text{ldb} \geq \text{kLocal}$ .
- **beta** – [in] Scaling of  $C$  before summation.
- **C** – [out] Pointer to global matrix  $C$ .
- **ldc** – [in] Leading dimension of  $C$  with  $\text{ldc} \geq \text{loc}(m)$ , where  $\text{loc}(m)$  is the number of locally stored rows of  $C$ .
- **cRowOffset** – [in] Row offset in the global matrix  $C$ , identifying the first row of the submatrix  $C$ .
- **cColOffset** – [in] Column offset in the global matrix  $C$ , identifying the first coloumn of the submatrix  $C$ .
- **distC** – [in] Matrix distribution of global matrix  $C$ .
- **ctx** – [in] *Context*, which provides configuration settings and reusable resources.

void **pgemm\_ssb**(int m, int n, int kLocal, *SplaOperation* opA, double alpha, const double \*A, int lda, const double \*B, int ldb, double beta, double \*C, int ldc, int cRowOffset, int cColOffset, *MatrixDistribution* &distC, *Context* &ctx)

Computes a distributed general matrix multiplication of the form  $C \leftarrow \alpha A^H B + \beta C$  in double precision.

See documentation above.

void **pgemm\_ssb**(int m, int n, int kLocal, *SplaOperation* opA, std::complex<float> alpha, const std::complex<float> \*A, int lda, const std::complex<float> \*B, int ldb, std::complex<float> beta, std::complex<float> \*C, int ldc, int cRowOffset, int cColOffset, *MatrixDistribution* &distC, *Context* &ctx)

Computes a distributed general matrix multiplication of the form  $C \leftarrow \alpha A^H B + \beta C$  in single precision for complex types.

See documentation above.

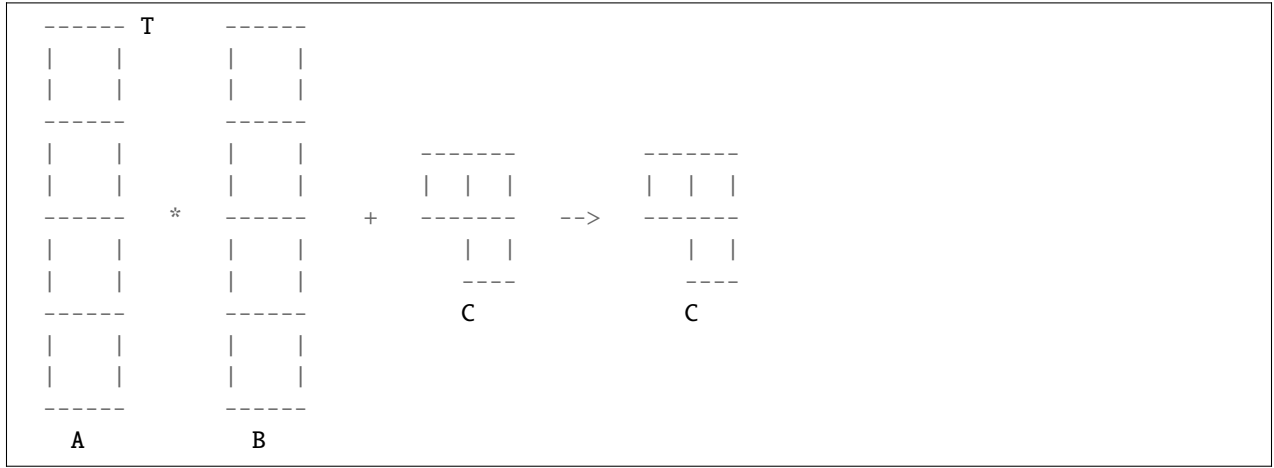
void **pgemm\_ssb**(int m, int n, int kLocal, *SplaOperation* opA, std::complex<double> alpha, const std::complex<double> \*A, int lda, const std::complex<double> \*B, int ldb, std::complex<double> beta, std::complex<double> \*C, int ldc, int cRowOffset, int cColOffset, *MatrixDistribution* &distC, *Context* &ctx)

Computes a distributed general matrix multiplication of the form  $C \leftarrow \alpha A^H B + \beta C$  in double precision for complex types.

See documentation above.

## GEMM - SSBTR

General matrix multiplication functions for computing  $C \leftarrow \alpha A^H B + \beta C$  with stripe-stripe-block distribution, where computation may be limited to triangular part.



### Functions

void **pgemm\_ssbtr**(int m, int n, int kLocal, *SplaOperation* opA, float alpha, const float \*A, int lda, const float \*B, int ldb, float beta, float \*C, int ldc, int cRowOffset, int cColOffset, *SplaFillMode* cFillMode, *MatrixDistribution* &distC, *Context* &ctx)

Computes a distributed general matrix multiplication of the form  $C \leftarrow \alpha A^H B + \beta C$  in single precision.

$A$  and  $B$  are only split along the row dimension (stripes), while  $C$  can be distributed as any supported *MatrixDistribution* type. The fill mode of  $C$  indicates the part of the matrix which must be computed, while any other part may or may not be computed. It is therefore not a strict limitation. For example, given `SPLA_FILL_MODE_UPPER`, a small matrix may still be fully computed, while a large matrix will be computed block wise, such that the computed blocks cover the upper triangle. The fill mode is always in reference to the full matrix, so offsets are taken into account.

#### Parameters

- **m** – [in] Number of rows of  $A^H$
- **n** – [in] Number of columns of  $B$
- **kLocal** – [in] Number rows of  $B$  and number of columns of  $A^H$  stored at calling MPI rank. This number may differ for each rank.
- **opA** – [in] Operation applied when reading matrix A. Must be `SPLA_OP_TRANSPOSE` or `SPLA_OP_CONJ_TRANSPOSE`.

- **alpha** – [in] Scaling of multiplication of  $A^H$  and  $B$
- **A** – [in] Pointer to matrix  $A$ .
- **lda** – [in] Leading dimension of  $A$  with  $lda \geq kLocal$ .
- **B** – [in] Pointer to matrix  $B$ .
- **ldb** – [in] Leading dimension of  $B$  with  $ldb \geq kLocal$ .
- **beta** – [in] Scaling of  $C$  before summation.
- **C** – [out] Pointer to global matrix  $C$ .
- **ldc** – [in] Leading dimension of  $C$  with  $ldc \geq loc(m)$ , where  $loc(m)$  is the number of locally stored rows of  $C$ .
- **cRowOffset** – [in] Row offset in the global matrix  $C$ , identifying the first row of the sub-matrix  $C$ .
- **cColOffset** – [in] Column offset in the global matrix  $C$ , identifying the first coloumn of the submatrix  $C$ .
- **cFillMode** – [in] Fill mode of matrix  $C$ .
- **distC** – [in] Matrix distribution of global matrix  $C$ .
- **ctx** – [in] *Context*, which provides configuration settings and reusable resources.

void **pgemm\_ssbtr**(int m, int n, int kLocal, *SplaOperation* opA, double alpha, const double \*A, int lda, const double \*B, int ldb, double beta, double \*C, int ldc, int cRowOffset, int cColOffset, *SplaFillMode* cFillMode, *MatrixDistribution* &distC, *Context* &ctx)

Computes a distributed general matrix multiplication of the form  $C \leftarrow \alpha A^H B + \beta C$  in double precision.

See documentation above.

void **pgemm\_ssbtr**(int m, int n, int kLocal, *SplaOperation* opA, std::complex<float> alpha, const std::complex<float> \*A, int lda, const std::complex<float> \*B, int ldb, std::complex<float> beta, std::complex<float> \*C, int ldc, int cRowOffset, int cColOffset, *SplaFillMode* cFillMode, *MatrixDistribution* &distC, *Context* &ctx)

Computes a distributed general matrix multiplication of the form  $C \leftarrow \alpha A^H B + \beta C$  in single precision for complex types.

See documentation above.

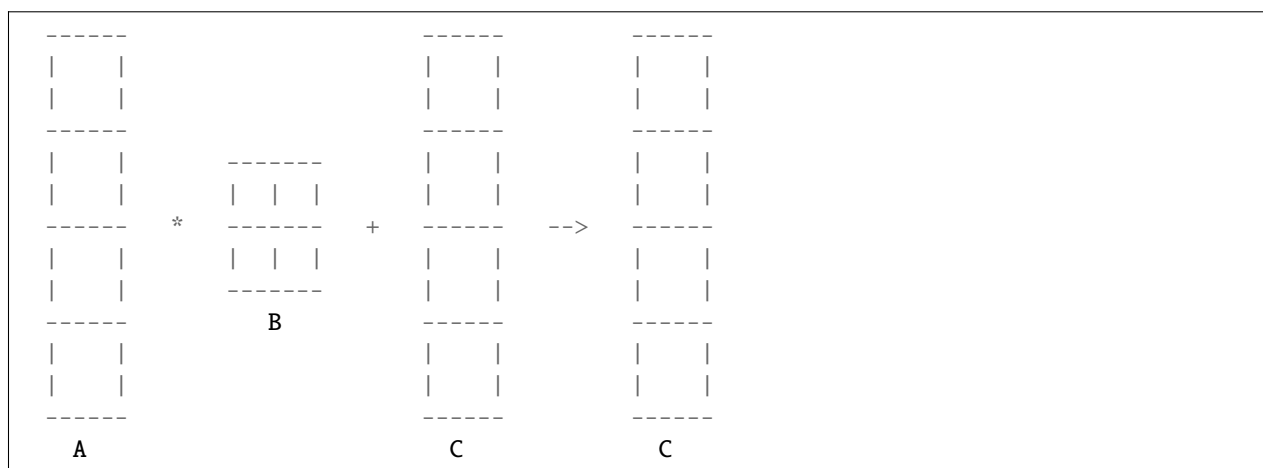
void **pgemm\_ssbtr**(int m, int n, int kLocal, *SplaOperation* opA, std::complex<double> alpha, const std::complex<double> \*A, int lda, const std::complex<double> \*B, int ldb, std::complex<double> beta, std::complex<double> \*C, int ldc, int cRowOffset, int cColOffset, *SplaFillMode* cFillMode, *MatrixDistribution* &distC, *Context* &ctx)

Computes a distributed general matrix multiplication of the form  $C \leftarrow \alpha A^H B + \beta C$  in double precision for complex types.

See documentation above.

## GEMM - SBS

General matrix multiplication functions for computing  $C \leftarrow \alpha AB + \beta C$  with stripe-block-stipe distribution.



### Functions

void **pgemm\_sbs**(int mLocal, int n, int k, float alpha, const float \*A, int lda, const float \*B, int ldb, int bRowOffset, int bColOffset, *MatrixDistribution* &distB, float beta, float \*C, int ldc, *Context* &ctx)

Computes a distributed general matrix multiplication of the form  $C \leftarrow \alpha AB + \beta C$  in single precision.

$A$  and  $C$  are only split along the row dimension (stripes), while  $B$  can be distributed as any supported *MatrixDistribution* type.

#### Parameters

- **mLocal** – [in] Number rows of  $A$  and  $C$  stored at calling MPI rank. This number may differ for each rank.
- **n** – [in] Number of columns of  $B$ .
- **k** – [in] Number of columns of  $C$  and rows of  $B$ .
- **alpha** – [in] Scaling of multiplication of  $A^H$  and  $B$
- **A** – [in] Pointer to matrix  $A$ .
- **lda** – [in] Leading dimension of  $A$  with  $lda \geq kLocal$ .
- **B** – [in] Pointer to matrix  $B$ .
- **ldb** – [in] Leading dimension of  $B$  with  $ldb \geq loc(k)$ , where  $loc(k)$  is the number of locally stored rows of  $B$ .

- **bRowOffset** – [in] Row offset in the global matrix  $B$ , identifying the first row of the sub-matrix  $B$ .
- **bColOffset** – [in] Column offset in the global matrix  $B$ , identifying the first coloumn of the submatrix  $B$ .
- **distB** – [in] Matrix distribution of global matrix  $B$ .
- **beta** – [in] Scaling of  $C$  before summation.
- **C** – [out] Pointer to matrix  $C$ .
- **ldc** – [in] Leading dimension of  $C$  with  $\text{ldC} \geq \text{mLocal}$ .
- **ctx** – [in] *Context*, which provides configuration settings and reusable resources.

void **pgemm\_sbs**(int mLocal, int n, int k, double alpha, const double \*A, int lda, const double \*B, int ldb, int bRowOffset, int bColOffset, *MatrixDistribution* &distB, double beta, double \*C, int ldc, *Context* &ctx)

Computes a distributed general matrix multiplication of the form  $C \leftarrow \alpha AB + \beta C$  in double precision.

See documentation above.

void **pgemm\_sbs**(int mLocal, int n, int k, std::complex<float> alpha, const std::complex<float> \*A, int lda, const std::complex<float> \*B, int ldb, int bRowOffset, int bColOffset, *MatrixDistribution* &distB, std::complex<float> beta, std::complex<float> \*C, int ldc, *Context* &ctx)

Computes a distributed general matrix multiplication of the form  $C \leftarrow \alpha AB + \beta C$  in double precision.

See documentation above.

void **pgemm\_sbs**(int mLocal, int n, int k, std::complex<double> alpha, const std::complex<double> \*A, int lda, const std::complex<double> \*B, int ldb, int bRowOffset, int bColOffset, *MatrixDistribution* &distB, std::complex<double> beta, std::complex<double> \*C, int ldc, *Context* &ctx)

Computes a distributed general matrix multiplication of the form  $C \leftarrow \alpha AB + \beta C$  in double precision.

See documentation above.



## EXCEPTIONS

namespace **spla**

class **GenericError** : public exception

*#include <exceptions.hpp>* A generic error.

Base type for all other exceptions.

Subclassed by *spla::GPUError*, *spla::InternalError*, *spla::InvalidAllocatorFunctionError*,  
*spla::InvalidParameterError*, *spla::InvalidPointerError*, *spla::MPIError*

### Public Functions

inline const char \***what**() const noexcept override

inline virtual *SplaError* **error\_code**() const noexcept

class **GPUAllocationError** : public *spla::GPUError*

*#include <exceptions.hpp>*

### Public Functions

inline const char \***what**() const noexcept override

inline virtual *SplaError* **error\_code**() const noexcept override

class **GPUBlasError** : public *spla::GPUError*

*#include <exceptions.hpp>*

### Public Functions

inline const char \***what**() const noexcept override

inline virtual *SplaError* **error\_code**() const noexcept override

class **GPUError** : public *spla::GenericError*

*#include <exceptions.hpp>* Subclassed by *spla::GPUAllocationError*, *spla::GPUBlasError*,  
*spla::GPUInvalidDevicePointerError*, *spla::GPUInvalidValueError*, *spla::GPULaunchError*,  
*spla::GPUNoDeviceError*, *spla::GPUSupportError*

### Public Functions

inline const char \***what**() const noexcept override

inline virtual *SplaError* **error\_code**() const noexcept override

```
class GPUInvalidDevicePointerError : public spla::GPUError  
    #include <exceptions.hpp>
```

### Public Functions

inline const char \***what**() const noexcept override

inline virtual *SplaError* **error\_code**() const noexcept override

```
class GPUInvalidValueError : public spla::GPUError  
    #include <exceptions.hpp>
```

### Public Functions

inline const char \***what**() const noexcept override

inline virtual *SplaError* **error\_code**() const noexcept override

```
class GPULaunchError : public spla::GPUError  
    #include <exceptions.hpp>
```

### Public Functions

inline const char \***what**() const noexcept override

inline virtual *SplaError* **error\_code**() const noexcept override

```
class GPUNoDeviceError : public spla::GPUError  
    #include <exceptions.hpp>
```

### Public Functions

inline const char \***what**() const noexcept override

inline virtual *SplaError* **error\_code**() const noexcept override

```
class GPUSupportError : public spla::GPUError  
    #include <exceptions.hpp>
```

### Public Functions

inline const char \***what**() const noexcept override

inline virtual *SplaError* **error\_code**() const noexcept override

class **InternalError** : public *spla::GenericError*  
*#include <exceptions.hpp>* Internal error.

### Public Functions

inline const char \***what**() const noexcept override

inline virtual *SplaError* **error\_code**() const noexcept override

class **InvalidAllocatorFunctionError** : public *spla::GenericError*  
*#include <exceptions.hpp>* Invalid allocator function error.

### Public Functions

inline const char \***what**() const noexcept override

inline virtual *SplaError* **error\_code**() const noexcept override

class **InvalidParameterError** : public *spla::GenericError*  
*#include <exceptions.hpp>* Invalid parameter error.

### Public Functions

inline const char \***what**() const noexcept override

inline virtual *SplaError* **error\_code**() const noexcept override

class **InvalidPointerError** : public *spla::GenericError*  
*#include <exceptions.hpp>* Invalid pointer error.

### Public Functions

inline const char \***what**() const noexcept override

inline virtual *SplaError* **error\_code**() const noexcept override

class **MPIAllocError** : public *spla::MPIError*  
*#include <exceptions.hpp>*

### Public Functions

inline const char \***what**() const noexcept override

inline virtual *SplaError* **error\_code**() const noexcept override

class **MPIError** : public *spla::GenericError*

*#include <exceptions.hpp>* Generic MPI Error.

Subclassed by *spla::MPIAllocError*, *spla::MPIThreadSupportError*

### Public Functions

inline const char \***what**() const noexcept override

inline virtual *SplaError* **error\_code**() const noexcept override

class **MPIThreadSupportError** : public *spla::MPIError*

*#include <exceptions.hpp>*

### Public Functions

inline const char \***what**() const noexcept override

inline virtual *SplaError* **error\_code**() const noexcept override

## CONTEXT

## Typedefs

typedef void \***SplaContext**

*Context* handle.

## Functions

*SplaError* **spla\_ctx\_create**(*SplaContext* \*ctx, *SplaProcessingUnit* pu)

Create *Context* with default configuration for given processing unit.

**Parameters**

- **ctx** – [out] *Context* handle.
- **pu** – [in] Processing unit to be used for computations.

**Returns**

Error code or SPLA\_SUCCESS.

*SplaError* **spla\_ctx\_destroy**(*SplaContext* \*ctx)

Destroy context.

**Parameters**

**ctx** – [in] *Context* handle.

**Returns**

Error code or SPLA\_SUCCESS.

*SplaError* **spla\_ctx\_processing\_unit**(*SplaContext* ctx, *SplaProcessingUnit* \*pu)

Access a *Context* parameter.

**Parameters**

- **ctx** – [in] *Context* handle.
- **pu** – [out] Processing unit used for computations.

**Returns**

Error code or SPLA\_SUCCESS.

**SPLA\_DEPRECATED** *SplaError* **spla\_ctx\_num\_threads** (*SplaContext* ctx, int \*numThreads)

Access a *Context* parameter.

**Parameters**

- **ctx** – [in] *Context* handle.
- **numThreads** – [out] Maximum number of threads used for computations.

**Returns**

Error code or SPLA\_SUCCESS.

*SplaError* **spla\_ctx\_num\_tiles**(*SplaContext* ctx, int \*numTiles)

Access a *Context* parameter.

**Parameters**

- **ctx** – [in] *Context* handle.
- **numTiles** – [out] Number of tiles used to overlap computation and communication.

**Returns**

Error code or SPLA\_SUCCESS.

*SplaError* **spla\_ctx\_tile\_size\_host**(*SplaContext* ctx, int \*tileSizeHost)

Access a *Context* parameter.

**Parameters**

- **ctx** – [in] *Context* handle.
- **tileSizeHost** – [out] Size of tiles for host computations and partitioning of communication.

**Returns**

Error code or SPLA\_SUCCESS.

*SplaError* **spla\_ctx\_tile\_size\_gpu**(*SplaContext* ctx, int \*tileSizeGPU)

Access a *Context* parameter.

**Parameters**

- **ctx** – [in] *Context* handle.
- **tileSizeGPU** – [out] Size of tiles on GPU.

**Returns**

Error code or SPLA\_SUCCESS.

*SplaError* **spla\_ctx\_op\_threshold\_gpu**(*SplaContext* ctx, int \*opThresholdGPU)

Access a *Context* parameter.

**Parameters**

- **ctx** – [in] *Context* handle.
- **opThresholdGPU** – [out] Operations threshold, below which computation may be done on Host, even if processing unit is set to GPU. For GEMM, the number of operations is estimated as  $2mnk$ .

**Returns**

Error code or SPLA\_SUCCESS.

*SplaError* **spla\_ctx\_gpu\_device\_id**(*SplaContext* ctx, int \*deviceId)

Access a *Context* parameter.

**Parameters**

- **ctx** – [in] *Context* handle.
- **deviceId** – [out] Id of GPU used for computations. This is set as fixed parameter by query of device id at context creation.

**Returns**

Error code or SPLA\_SUCCESS.

*SplaError* **spla\_ctx\_allocated\_memory\_host**(*SplaContext* ctx, uint\_least64\_t \*size)

Access a *Context* parameter.

**Parameters**

- **ctx** – [in] *Context* handle.
- **size** – [in] Total allocated memory on host in bytes used for internal buffers. Does not include allocations through standard C++ allocators. May change with with use of context.

**Returns**

Error code or SPLA\_SUCCESS.

*SplaError* **spla\_ctx\_allocated\_memory\_pinned**(*SplaContext* ctx, uint\_least64\_t \*size)

Access a *Context* parameter.

**Parameters**

- **ctx** – [in] *Context* handle.
- **size** – [in] Total allocated pinned memory on host in bytes used for internal buffers. Does not include allocations through standard C++ allocators. May change with with use of context.

**Returns**

Error code or SPLA\_SUCCESS.

*SplaError* **spla\_ctx\_allocated\_memory\_gpu**(*SplaContext* ctx, uint\_least64\_t \*size)

Access a *Context* parameter.

**Parameters**

- **ctx** – [in] *Context* handle.
- **size** – [in] Total allocated memory on gpu in bytes used for internal buffers. Does not include allocations by device libraries like cuBLAS / rocBLAS. May change with with use of context.

**Returns**

Error code or SPLA\_SUCCESS.

**SPLA\_DEPRECATED** *SplaError* **spla\_ctx\_set\_num\_threads** (*SplaContext* ctx, int numThreads)

Set the number of threads to be used.

**Parameters**

- **ctx** – [in] *Context* handle.
- **numThreads** – [in] Number of threads.

**Returns**

Error code or SPLA\_SUCCESS.

*SplaError* **spla\_ctx\_set\_num\_tiles**(*SplaContext* ctx, int numTiles)

Set the number of tiles.

**Parameters**

- **ctx** – [in] *Context* handle.
- **numTiles** – [in] Number of tiles.

**Returns**

Error code or SPLA\_SUCCESS.

*SplaError* **spla\_ctx\_set\_tile\_size\_host**(*SplaContext* ctx, int tileSizeHost)

Set the tile size used for computations on host and partitioning communication.

**Parameters**

- **ctx** – [in] *Context* handle.
- **tileSizeHost** – [in] Size of tiles on host.

**Returns**

Error code or SPLA\_SUCCESS.

*SplaError* **spla\_ctx\_set\_op\_threshold\_gpu**(*SplaContext* ctx, int opThresholdGPU)

Set the operations threshold, below which computation may be done on Host, even if processing unit is set to GPU.

For GEMM, the number of operations is estimated as  $2mnk$ .

**Parameters**

- **ctx** – [in] *Context* handle.
- **opThresholdGPU** – [in] Threshold in number of operations.

**Returns**

Error code or SPLA\_SUCCESS.

*SplaError* **spla\_ctx\_set\_tile\_size\_gpu**(*SplaContext* ctx, int tileSizeGPU)

Set tile size for GPU computations.

**Parameters**

- **ctx** – [in] *Context* handle.
- **tileSizeGPU** – [in] Size of tiles on GPU.

**Returns**

Error code or SPLA\_SUCCESS.

*SplaError* **spla\_ctx\_set\_alloc\_host**(*SplaContext* ctx, void \*(\*allocateFunc)(size\_t), void (\*deallocateFunc)(void\*))

Set the allocation and deallocation functions for host memory.

Internal default uses a memory pool for better performance. Not available in Fortran module.

**Parameters**

- **ctx** – [in] *Context* handle.
- **allocateFunc** – [in] Function allocating given size in bytes.
- **deallocateFunc** – [in] Function to deallocate memory allocated using allocateFunc.

**Returns**

Error code or SPLA\_SUCCESS.

*SplaError* **spla\_ctx\_set\_alloc\_pinned**(*SplaContext* ctx, void \*(\*allocateFunc)(size\_t), void (\*deallocateFunc)(void\*))

Set the allocation and deallocation functions for pinned host memory.

Internal default uses a memory pool for better performance. Not available in Fortran module.

**Parameters**



- **ctx** – [in] *Context* handle.
- **allocateFunc** – [in] Function allocating given size in bytes.
- **deallocateFunc** – [in] Function to deallocate memory allocated using allocateFunc.

**Returns**

Error code or SPLA\_SUCCESS.

*SplaError* **spla\_ctx\_set\_alloc\_gpu**(*SplaContext* ctx, void \*(\*allocateFunc)(size\_t), void  
(\*deallocateFunc)(void\*))

Set the allocation and deallocation functions for gpu memory.

Internal default uses a memory pool for better performance. Not available in Fortran module.

**Parameters**

- **ctx** – [in] *Context* handle.
- **allocateFunc** – [in] Function allocating given size in bytes.
- **deallocateFunc** – [in] Function to deallocate memory allocated using allocateFunc.

**Returns**

Error code or SPLA\_SUCCESS.



## MATRIX DISTRIBUTION

### Typedefs

typedef void \***SplaMatrixDistribution**  
Matrix distribution handle.

### Functions

*SplaError* **spla\_mat\_dis\_create\_block\_cyclic**(*SplaMatrixDistribution* \*matDis, MPI\_Comm comm, char order, int procGridRows, int procGridCols, int rowBlockSize, int colBlockSize)

Create a blacs block cyclic matrix distribution with row major or coloumn major ordering of MPI ranks.

#### Parameters

- **matDis** – [out] Matrix distribution handle.
- **comm** – [in] MPI communicator to be used.
- **order** – [in] Either 'R' for row major ordering or 'C' for coloumn major ordering.
- **procGridRows** – [in] Number of rows in process grid.
- **procGridCols** – [in] Number of coloumns in process grid.
- **rowBlockSize** – [in] Row block size for matrix partitioning.
- **colBlockSize** – [in] Coloumn block size for matrix partitioning.

#### Returns

Error code or SPLA\_SUCCESS.

*SplaError* **spla\_mat\_dis\_create\_blacs\_block\_cyclic\_from\_mapping**(*SplaMatrixDistribution* \*matDis, MPI\_Comm comm, const int \*mapping, int procGridRows, int procGridCols, int rowBlockSize, int colBlockSize)

Create a blacs block cyclic matrix distribution with given process grid mapping.

#### Parameters

- **matDis** – [out] Matrix distribution handle.
- **comm** – [in] MPI communicator to be used.
- **mapping** – [in] Pointer to array of size procGridRows \* procGridCols mapping MPI ranks onto a coloumn major process grid.

- **procGridRows** – [in] Number of rows in process grid.
- **procGridCols** – [in] Number of columns in process grid.
- **rowBlockSize** – [in] Row block size for matrix partitioning.
- **colBlockSize** – [in] Column block size for matrix partitioning.

**Returns**

Error code or SPLA\_SUCCESS.

*SplaError* **spla\_mat\_dis\_destroy**(*SplaMatrixDistribution* \*matDis)

Destroy matrix distribution.

**Parameters**

**matDis** – [in] Matrix distribution handle.

**Returns**

Error code or SPLA\_SUCCESS.

*SplaError* **spla\_mat\_dis\_create\_mirror**(*SplaMatrixDistribution* \*matDis, MPI\_Comm comm)

Create a mirror distribution, where the full matrix is stored on each MPI rank.

**Parameters**

- **matDis** – [out] Matrix distribution handle.
- **comm** – [in] MPI communicator to be used.

**Returns**

Error code or SPLA\_SUCCESS.

*SplaError* **spla\_mat\_dis\_proc\_grid\_rows**(*SplaMatrixDistribution* matDis, int \*procGridRows)

Access a distribution parameter.

**Parameters**

- **matDis** – [in] Matrix distribution handle.
- **procGridRows** – [out] Number of rows in process grid.

**Returns**

Error code or SPLA\_SUCCESS.

*SplaError* **spla\_mat\_dis\_proc\_grid\_cols**(*SplaMatrixDistribution* matDis, int \*procGridCols)

Access a distribution parameter.

**Parameters**

- **matDis** – [in] Matrix distribution handle.
- **procGridCols** – [out] Number of columns in process grid.

**Returns**

Error code or SPLA\_SUCCESS.

*SplaError* **spla\_mat\_dis\_row\_block\_size**(*SplaMatrixDistribution* matDis, int \*rowBlockSize)

Access a distribution parameter.

**Parameters**

- **matDis** – [in] Matrix distribution handle.
- **rowBlockSize** – [out] Row block size used for matrix partitioning.

**Returns**

Error code or SPLA\_SUCCESS.

*SplaError* **spla\_mat\_dis\_col\_block\_size**(*SplaMatrixDistribution* matDis, int \*colBlockSize)

Access a distribution parameter.

**Parameters**

- **matDis** – [in] Matrix distribution handle.
- **colBlockSize** – [out] Coloumn block size used for matrix partitioning.

**Returns**

Error code or SPLA\_SUCCESS.

*SplaError* **spla\_mat\_dis\_type**(*SplaMatrixDistribution* matDis, *SplaDistributionType* \*type)

Access a distribution parameter.

**Parameters**

- **matDis** – [in] Matrix distribution handle.
- **type** – [out] Distribution type

**Returns**

Error code or SPLA\_SUCCESS.

*SplaError* **spla\_mat\_dis\_comm**(*SplaMatrixDistribution* matDis, MPI\_Comm \*comm)

Access a distribution parameter.

**Parameters**

- **matDis** – [in] Matrix distribution handle.
- **comm** – [out] Communicator used internally. Order of ranks may differ from communicator provided for creation of distribution.

**Returns**

Error code or SPLA\_SUCCESS.

*SplaError* **spla\_mat\_dis\_set\_row\_block\_size**(*SplaMatrixDistribution* matDis, int rowBlockSize)

Set a distribution parameter.

**Parameters**

- **matDis** – [in] Matrix distribution handle.
- **rowBlockSize** – [in] Row block size used for matrix partitioning. provided for creation of distribution.

**Returns**

Error code or SPLA\_SUCCESS.

*SplaError* **spla\_mat\_dis\_set\_col\_block\_size**(*SplaMatrixDistribution* matDis, int colBlockSize)

Set a distribution parameter.

**Parameters**

- **matDis** – [in] Matrix distribution handle.
- **colBlockSize** – [in] Col block size used for matrix partitioning. provided for creation of distribution.

**Returns**

Error code or SPLA\_SUCCESS.



## GEMM

General matrix multiplication functions for locally computing  $C \leftarrow \alpha OP(A)OP(B) + \beta C$ .

### Functions

*SplaError* **spla\_sgemm**(*SplaOperation* opA, *SplaOperation* opB, int m, int n, int k, float alpha, const float \*A, int lda, const float \*B, int ldb, float beta, float \*C, int ldc, *SplaContext* ctx)

Computes a local general matrix multiplication of the form  $C \leftarrow \alpha op(A)op(B) + \beta C$  in single precision.

If context with processing unit set to GPU, pointers to matrices can be any combination of host and device pointers.

#### Parameters

- **opA** – [in] Operation applied when reading matrix  $A$ .
- **opB** – [in] Operation applied when reading matrix  $B$ .
- **m** – [in] Number of rows of  $OP(A)$
- **n** – [in] Number of columns of  $OP(B)$
- **k** – [in] Number rows of  $OP(B)$  and number of columns of  $OP(A)$
- **alpha** – [in] Scaling of multiplication of  $A^H$  and  $B$
- **A** – [in] Pointer to matrix  $A$ .
- **lda** – [in] Leading dimension of  $A$ .
- **B** – [in] Pointer to matrix  $B$ .
- **ldb** – [in] Leading dimension of  $B$ .
- **beta** – [in] Scaling of  $C$  before summation.
- **C** – [out] Pointer to matrix  $C$ .
- **ldc** – [in] Leading dimension of  $C$ .
- **ctx** – [in] *Context* handle, which provides configuration settings and reusable resources.

*SplaError* **spla\_dgemm**(*SplaOperation* opA, *SplaOperation* opB, int m, int n, int k, double alpha, const double \*A, int lda, const double \*B, int ldb, double beta, double \*C, int ldc, *SplaContext* ctx)

Computes a local general matrix multiplication of the form  $C \leftarrow \alpha OP(A)OP(B) + \beta C$  in double precision.

See documentation above.

*SplaError* **spla\_cgemm**(*SplaOperation* opA, *SplaOperation* opB, int m, int n, int k, const void \*alpha, const void \*A, int lda, const void \*B, int ldb, const void \*beta, void \*C, int ldc, *SplaContext* ctx)

Computes a local general matrix multiplication of the form  $C \leftarrow \alpha OP(A)OP(B) + \beta C$  in single precision complex types.

See documentation above.

*SplaError* **spla\_zgemm**(*SplaOperation* opA, *SplaOperation* opB, int m, int n, int k, const void \*alpha, const void \*A, int lda, const void \*B, int ldb, const void \*beta, void \*C, int ldc, *SplaContext* ctx)

Computes a local general matrix multiplication of the form  $C \leftarrow \alpha OP(A)OP(B) + \beta C$  in double precision complex types.

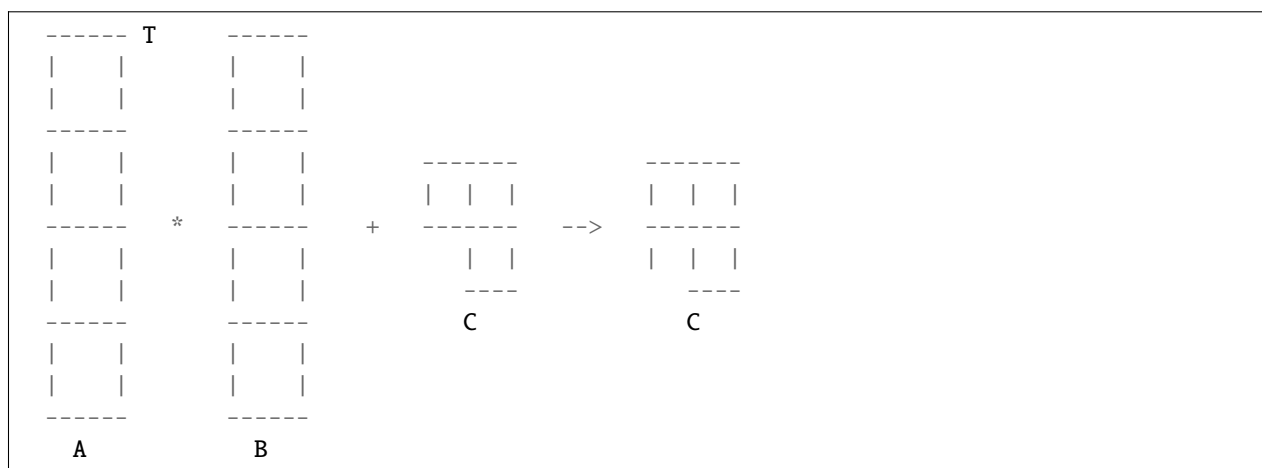
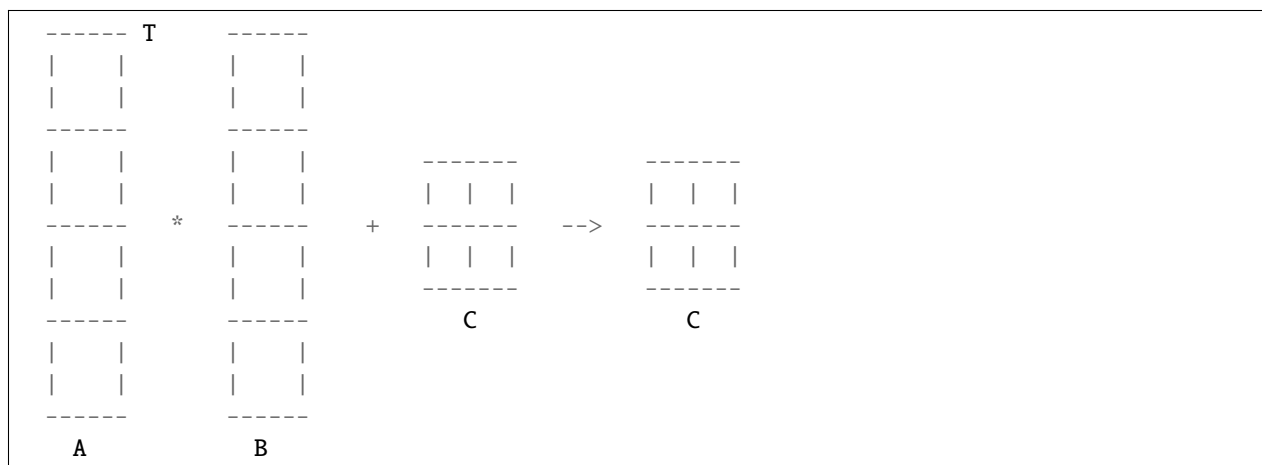
See documentation above.



## GEMM - SSB

General matrix multiplication functions for computing  $C \leftarrow \alpha A^H B + \beta C$  with stripe-stripe-block distribution.

General matrix multiplication functions for computing  $C \leftarrow \alpha A^H B + \beta C$  with stripe-stripe-block distribution, where computation may be limited to triangular part.



## Functions

*SplaError* **spla\_psgemm\_ssb**(int m, int n, int kLocal, *SplaOperation* opA, float alpha, const float \*A, int lda, const float \*B, int ldb, float beta, float \*C, int ldc, int cRowOffset, int cColOffset, *SplaMatrixDistribution* distC, *SplaContext* ctx)

Computes a distributed general matrix multiplication of the form  $C \leftarrow \alpha A^H B + \beta C$  in single precision.

$A$  and  $B$  are only split along the row dimension (stripes), while  $C$  can be distributed as any supported *SplaMatrixDistribution* type.

### Parameters

- **m** – [in] Number of rows of  $A^H$
- **n** – [in] Number of columns of  $B$
- **kLocal** – [in] Number rows of  $B$  and number of columns of  $A^H$  stored at calling MPI rank. This number may differ for each rank.
- **opA** – [in] Operation applied when reading matrix  $A$ . Must be `SPLA_OP_TRANSPOSE` or
- **alpha** – [in] Scaling of multiplication of  $A^H$  and  $B$
- **A** – [in] Pointer to matrix  $A$ .
- **lda** – [in] Leading dimension of  $A$  with  $lda \geq kLocal$ .
- **B** – [in] Pointer to matrix  $B$ .
- **ldb** – [in] Leading dimension of  $B$  with  $ldb \geq kLocal$ .
- **beta** – [in] Scaling of  $C$  before summation.
- **C** – [out] Pointer to global matrix  $C$ .
- **ldc** – [in] Leading dimension of  $C$  with  $ldc \geq loc(m)$ , where  $loc(m)$  is the number of locally stored rows of  $C$ .
- **cRowOffset** – [in] Row offset in the global matrix  $C$ , identifying the first row of the submatrix  $C$ .
- **cColOffset** – [in] Column offset in the global matrix  $C$ , identifying the first coloumn of the submatrix  $C$ .
- **distC** – [in] Matrix distribution of global matrix  $C$ .
- **ctx** – [in] *Context*, which provides configuration settings and reusable resources.

*SplaError* **spla\_pdgemm\_ssb**(int m, int n, int kLocal, *SplaOperation* opA, double alpha, const double \*A, int lda, const double \*B, int ldb, double beta, double \*C, int ldc, int cRowOffset, int cColOffset, *SplaMatrixDistribution* distC, *SplaContext* ctx)

Computes a distributed general matrix multiplication of the form  $C \leftarrow \alpha A^H B + \beta C$  in double precision.

See documentation above.

*SplaError* **spla\_pcgemm\_ssb**(int m, int n, int kLocal, *SplaOperation* opA, const void \*alpha, const void \*A, int lda, const void \*B, int ldb, const void \*beta, void \*C, int ldc, int cRowOffset, int cColOffset, *SplaMatrixDistribution* distC, *SplaContext* ctx)

Computes a distributed general matrix multiplication of the form  $C \leftarrow \alpha A^H B + \beta C$  in single precision for complex types.

See documentation above.

*SplaError* **spla\_pzgemm\_ssb**(int m, int n, int kLocal, *SplaOperation* opA, const void \*alpha, const void \*A, int lda, const void \*B, int ldb, const void \*beta, void \*C, int ldc, int cRowOffset, int cColOffset, *SplaMatrixDistribution* distC, *SplaContext* ctx)

Computes a distributed general matrix multiplication of the form  $C \leftarrow \alpha A^H B + \beta C$  in double precision for complex types.

See documentation above.



## GEMM - SSBTR

### Functions

*SplaError* **spla\_psgemm\_ssbtr**(int m, int n, int kLocal, *SplaOperation* opA, float alpha, const float \*A, int lda, const float \*B, int ldb, float beta, float \*C, int ldc, int cRowOffset, int cColOffset, *SplaFillMode* cFillMode, *SplaMatrixDistribution* distC, *SplaContext* ctx)

Computes a distributed general matrix multiplication of the form  $C \leftarrow \alpha A^H B + \beta C$  in single precision.

$A$  and  $B$  are only split along the row dimension (stripes), while  $C$  can be distributed as any supported *SplaMatrixDistribution* type. The fill mode of  $C$  indicates the part of the matrix which must be computed, while any other part may or may not be computed. It is therefore not a strict limitation. For example, given *SPLA\_FILL\_MODE\_UPPER*, a small matrix may still be fully computed, while a large matrix will be computed block wise, such that the computed blocks cover the upper triangle. The fill mode is always in reference to the full matrix, so offsets are taken into account.

#### Parameters

- **m** – [in] Number of rows of  $A^H$
- **n** – [in] Number of columns of  $B$
- **kLocal** – [in] Number rows of  $B$  and number of columns of  $A^H$  stored at calling MPI rank. This number may differ for each rank.
- **opA** – [in] Operation applied when reading matrix  $A$ . Must be *SPLA\_OP\_TRANSPOSE* or
- **alpha** – [in] Scaling of multiplication of  $A^H$  and  $B$
- **A** – [in] Pointer to matrix  $A$ .
- **lda** – [in] Leading dimension of  $A$  with  $lda \geq kLocal$ .
- **B** – [in] Pointer to matrix  $B$ .
- **ldb** – [in] Leading dimension of  $B$  with  $ldb \geq kLocal$ .
- **beta** – [in] Scaling of  $C$  before summation.
- **C** – [out] Pointer to global matrix  $C$ .
- **ldc** – [in] Leading dimension of  $C$  with  $ldc \geq loc(m)$ , where  $loc(m)$  is the number of locally stored rows of  $C$ .
- **cRowOffset** – [in] Row offset in the global matrix  $C$ , identifying the first row of the submatrix  $C$ .
- **cColOffset** – [in] Column offset in the global matrix  $C$ , identifying the first column of the submatrix  $C$ .
- **cFillMode** – [in] Fill mode of matrix  $C$ .

- **distC** – [in] Matrix distribution of global matrix  $C$ .
- **ctx** – [in] *Context*, which provides configuration settings and reusable resources.

*SplaError* **spla\_pdgemm\_ssbtr**(int m, int n, int kLocal, *SplaOperation* opA, double alpha, const double \*A, int lda, const double \*B, int ldb, double beta, double \*C, int ldc, int cRowOffset, int cColOffset, *SplaFillMode* cFillMode, *SplaMatrixDistribution* distC, *SplaContext* ctx)

Computes a distributed general matrix multiplication of the form  $C \leftarrow \alpha A^H B + \beta C$  in double precision.

See documentation above.

*SplaError* **spla\_pcgemm\_ssbtr**(int m, int n, int kLocal, *SplaOperation* opA, const void \*alpha, const void \*A, int lda, const void \*B, int ldb, const void \*beta, void \*C, int ldc, int cRowOffset, int cColOffset, *SplaFillMode* cFillMode, *SplaMatrixDistribution* distC, *SplaContext* ctx)

Computes a distributed general matrix multiplication of the form  $C \leftarrow \alpha A^H B + \beta C$  in single precision for complex types.

See documentation above.

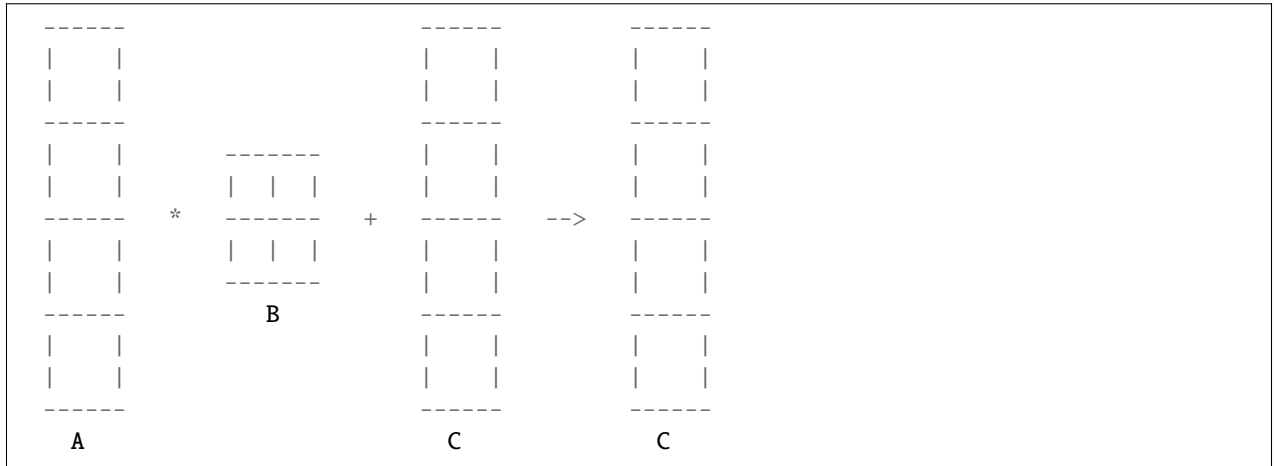
*SplaError* **spla\_pzgemm\_ssbtr**(int m, int n, int kLocal, *SplaOperation* opA, const void \*alpha, const void \*A, int lda, const void \*B, int ldb, const void \*beta, void \*C, int ldc, int cRowOffset, int cColOffset, *SplaFillMode* cFillMode, *SplaMatrixDistribution* distC, *SplaContext* ctx)

Computes a distributed general matrix multiplication of the form  $C \leftarrow \alpha A^H B + \beta C$  in double precision for complex types.

See documentation above.

## GEMM - SBS

General matrix multiplication functions for computing  $C \leftarrow \alpha AB + \beta C$  with stripe-block-stipe distribution.



### Functions

*SplaError* **spla\_psgemm\_sbs**(int mLocal, int n, int k, float alpha, const float \*A, int lda, const float \*B, int ldb, int bRowOffset, int bColOffset, *SplaMatrixDistribution* distB, float beta, float \*C, int ldc, *SplaContext* ctx)

Computes a distributed general matrix multiplication of the form  $C \leftarrow \alpha AB + \beta C$  in single precision.

$A$  and  $C$  are only split along the row dimension (stripes), while  $B$  can be distributed as any supported *MatrixDistribution* type.

#### Parameters

- **mLocal** – [in] Number rows of  $A$  and  $C$  stored at calling MPI rank. This number may differ for each rank.
- **n** – [in] Number of columns of  $B$ .
- **k** – [in] Number of columns of  $C$  and rows of  $B$ .
- **alpha** – [in] Scaling of multiplication of  $A^H$  and  $B$
- **A** – [in] Pointer to matrix  $A$ .
- **lda** – [in] Leading dimension of  $A$  with  $lda \geq kLocal$ .
- **B** – [in] Pointer to matrix  $B$ .

- **ldb** – [in] Leading dimension of  $B$  with  $\text{ldb} \geq \text{loc}(k)$ , where  $\text{loc}(k)$  is the number of locally stored rows of  $B$ .
- **bRowOffset** – [in] Row offset in the global matrix  $B$ , identifying the first row of the submatrix  $B$ .
- **bColOffset** – [in] Column offset in the global matrix  $B$ , identifying the first column of the submatrix  $B$ .
- **distB** – [in] Matrix distribution of global matrix  $B$ .
- **beta** – [in] Scaling of  $C$  before summation.
- **C** – [out] Pointer to matrix  $C$ .
- **ldc** – [in] Leading dimension of  $C$  with  $\text{ldc} \geq \text{mLocal}$ .
- **ctx** – [in] *Context*, which provides configuration settings and reusable resources.

*SplaError* **spla\_pdgemv\_sbs**(int mLocal, int n, int k, double alpha, const double \*A, int lda, const double \*B, int ldb, int bRowOffset, int bColOffset, *SplaMatrixDistribution* distB, double beta, double \*C, int ldc, *SplaContext* ctx)

Computes a distributed general matrix multiplication of the form  $C \leftarrow \alpha AB + \beta C$  in double precision.

See documentation above.

*SplaError* **spla\_pcgemv\_sbs**(int mLocal, int n, int k, const void \*alpha, const void \*A, int lda, const void \*B, int ldb, int bRowOffset, int bColOffset, *SplaMatrixDistribution* distB, const void \*beta, void \*C, int ldc, *SplaContext* ctx)

Computes a distributed general matrix multiplication of the form  $C \leftarrow \alpha AB + \beta C$  in double precision.

See documentation above.

*SplaError* **spla\_pzgemv\_sbs**(int mLocal, int n, int k, const void \*alpha, const void \*A, int lda, const void \*B, int ldb, int bRowOffset, int bColOffset, *SplaMatrixDistribution* distB, const void \*beta, void \*C, int ldc, *SplaContext* ctx)

Computes a distributed general matrix multiplication of the form  $C \leftarrow \alpha AB + \beta C$  in double precision.

See documentation above.



## ERRORS

### Enums

enum **SplaError**

*Values:*

enumerator **SPLA\_SUCCESS**

Success.

No error.

enumerator **SPLA\_UNKNOWN\_ERROR**

Unknown error.

enumerator **SPLA\_INTERNAL\_ERROR**

Internal error.

enumerator **SPLA\_INVALID\_PARAMETER\_ERROR**

Invalid parameter error.

enumerator **SPLA\_INVALID\_POINTER\_ERROR**

Invalid pointer error.

enumerator **SPLA\_INVALID\_HANDLE\_ERROR**

Invalid handle error.

enumerator **SPLA\_MPI\_ERROR**

MPI error.

enumerator **SPLA\_MPI\_ALLOCATION\_ERROR**

MPI allocation error.

enumerator **SPLA\_MPI\_THREAD\_SUPPORT\_ERROR**

MPI thread support error.

enumerator **SPLA\_GPU\_ERROR**

GPU error.

enumerator **SPLA\_GPU\_SUPPORT\_ERROR**

GPU support error.

enumerator **SPLA\_GPU\_ALLOCATION\_ERROR**

GPU allocation error.

enumerator **SPLA\_GPU\_LAUNCH\_ERROR**

GPU launch error.

enumerator **SPLA\_GPU\_NO\_DEVICE\_ERROR**

GPU no device error.

enumerator **SPLA\_GPU\_INVALID\_VALUE\_ERROR**

GPU invalid value error.

enumerator **SPLA\_GPU\_INVALID\_DEVICE\_POINTER\_ERROR**

Invalid device pointer error.

enumerator **SPLA\_GPU\_BLAS\_ERROR**

GPU blas error.

enumerator **SPLA\_INVALID\_ALLOCATOR\_FUNCTION**

Invalid allocator function error.

## C

Context (C++ class), 5  
 Context::allocated\_memory\_gpu (C++ function), 6  
 Context::allocated\_memory\_host (C++ function), 6  
 Context::allocated\_memory\_pinned (C++ function), 6  
 Context::Context (C++ function), 5  
 Context::gpu\_device\_id (C++ function), 6  
 Context::num\_tiles (C++ function), 5  
 Context::op\_threshold\_gpu (C++ function), 6  
 Context::operator= (C++ function), 5  
 Context::processing\_unit (C++ function), 5  
 Context::set\_alloc\_gpu (C++ function), 7  
 Context::set\_alloc\_host (C++ function), 7  
 Context::set\_alloc\_pinned (C++ function), 7  
 Context::set\_num\_tiles (C++ function), 6  
 Context::set\_op\_threshold\_gpu (C++ function), 7  
 Context::set\_tile\_size\_gpu (C++ function), 7  
 Context::set\_tile\_size\_host (C++ function), 6  
 Context::tile\_size\_gpu (C++ function), 6  
 Context::tile\_size\_host (C++ function), 5

## G

gemm (C++ function), 13, 14

## M

MatrixDistribution (C++ class), 9  
 MatrixDistribution::col\_block\_size (C++ function), 9  
 MatrixDistribution::comm (C++ function), 10  
 MatrixDistribution::create\_blacs\_block\_cyclic (C++ function), 10  
 MatrixDistribution::create\_blacs\_block\_cyclic\_from\_mapping (C++ function), 10  
 MatrixDistribution::create\_mirror (C++ function), 11  
 MatrixDistribution::MatrixDistribution (C++ function), 9  
 MatrixDistribution::operator= (C++ function), 9  
 MatrixDistribution::proc\_grid\_cols (C++ function), 9

MatrixDistribution::proc\_grid\_rows (C++ function), 9  
 MatrixDistribution::row\_block\_size (C++ function), 9  
 MatrixDistribution::set\_col\_block\_size (C++ function), 10  
 MatrixDistribution::set\_row\_block\_size (C++ function), 10  
 MatrixDistribution::type (C++ function), 9

## P

pgemmm\_sbs (C++ function), 19, 20  
 pgemmm\_ssb (C++ function), 15, 16  
 pgemmm\_ssbtr (C++ function), 17, 18

## S

spla (C++ type), 21  
 spla::GenericError (C++ class), 21  
 spla::GenericError::error\_code (C++ function), 21  
 spla::GenericError::what (C++ function), 21  
 spla::GPUAllocationError (C++ class), 21  
 spla::GPUAllocationError::error\_code (C++ function), 21  
 spla::GPUAllocationError::what (C++ function), 21  
 spla::GPUBlasError (C++ class), 21  
 spla::GPUBlasError::error\_code (C++ function), 21  
 spla::GPUBlasError::what (C++ function), 21  
 spla::GPUError (C++ class), 21  
 spla::GPUError::error\_code (C++ function), 22  
 spla::GPUError::what (C++ function), 22  
 spla::GPUInvalidDevicePointerError (C++ class), 22  
 spla::GPUInvalidDevicePointerError::error\_code (C++ function), 22  
 spla::GPUInvalidDevicePointerError::what (C++ function), 22  
 spla::GPUInvalidValueError (C++ class), 22  
 spla::GPUInvalidValueError::error\_code (C++ function), 22

```

spla::GPUInvalidValueError::what (C++ function), 22
spla::GPULaunchError (C++ class), 22
spla::GPULaunchError::error_code (C++ function), 22
spla::GPULaunchError::what (C++ function), 22
spla::GPUNoDeviceError (C++ class), 22
spla::GPUNoDeviceError::error_code (C++ function), 22
spla::GPUNoDeviceError::what (C++ function), 22
spla::GPUSupportError (C++ class), 22
spla::GPUSupportError::error_code (C++ function), 23
spla::GPUSupportError::what (C++ function), 23
spla::InternalError (C++ class), 23
spla::InternalError::error_code (C++ function), 23
spla::InternalError::what (C++ function), 23
spla::InvalidAllocatorFunctionError (C++ class), 23
spla::InvalidAllocatorFunctionError::error_code (C++ function), 23
spla::InvalidAllocatorFunctionError::what (C++ function), 23
spla::InvalidParameterError (C++ class), 23
spla::InvalidParameterError::error_code (C++ function), 23
spla::InvalidParameterError::what (C++ function), 23
spla::InvalidPointerError (C++ class), 23
spla::InvalidPointerError::error_code (C++ function), 23
spla::InvalidPointerError::what (C++ function), 23
spla::MPIAllocError (C++ class), 23
spla::MPIAllocError::error_code (C++ function), 24
spla::MPIAllocError::what (C++ function), 24
spla::MPIError (C++ class), 24
spla::MPIError::error_code (C++ function), 24
spla::MPIError::what (C++ function), 24
spla::MPIThreadSupportError (C++ class), 24
spla::MPIThreadSupportError::error_code (C++ function), 24
spla::MPIThreadSupportError::what (C++ function), 24
spla_cgemm (C++ function), 35
spla_ctx_allocated_memory_gpu (C++ function), 27
spla_ctx_allocated_memory_host (C++ function), 27
spla_ctx_allocated_memory_pinned (C++ function), 27
spla_ctx_create (C++ function), 25
spla_ctx_destroy (C++ function), 25
spla_ctx_gpu_device_id (C++ function), 26
spla_ctx_num_tiles (C++ function), 26
spla_ctx_op_threshold_gpu (C++ function), 26
spla_ctx_processing_unit (C++ function), 25
spla_ctx_set_alloc_gpu (C++ function), 29
spla_ctx_set_alloc_host (C++ function), 28
spla_ctx_set_alloc_pinned (C++ function), 28
spla_ctx_set_num_tiles (C++ function), 27
spla_ctx_set_op_threshold_gpu (C++ function), 28
spla_ctx_set_tile_size_gpu (C++ function), 28
spla_ctx_set_tile_size_host (C++ function), 28
spla_ctx_tile_size_gpu (C++ function), 26
spla_ctx_tile_size_host (C++ function), 26
spla_dgemm (C++ function), 35
spla_mat_dis_col_block_size (C++ function), 33
spla_mat_dis_comm (C++ function), 33
spla_mat_dis_create_blacs_block_cyclic_from_mapping (C++ function), 31
spla_mat_dis_create_block_cyclic (C++ function), 31
spla_mat_dis_create_mirror (C++ function), 32
spla_mat_dis_destroy (C++ function), 32
spla_mat_dis_proc_grid_cols (C++ function), 32
spla_mat_dis_proc_grid_rows (C++ function), 32
spla_mat_dis_row_block_size (C++ function), 32
spla_mat_dis_set_col_block_size (C++ function), 33
spla_mat_dis_set_row_block_size (C++ function), 33
spla_mat_dis_type (C++ function), 33
spla_pcgemm_sbs (C++ function), 44
spla_pcgemm_ssb (C++ function), 38
spla_pcgemm_ssbtr (C++ function), 42
spla_pdgemm_sbs (C++ function), 44
spla_pdgemm_ssb (C++ function), 38
spla_pdgemm_ssbtr (C++ function), 42
spla_psgemm_sbs (C++ function), 43
spla_psgemm_ssb (C++ function), 38
spla_psgemm_ssbtr (C++ function), 41
spla_pzgemm_sbs (C++ function), 44
spla_pzgemm_ssb (C++ function), 38
spla_pzgemm_ssbtr (C++ function), 42
spla_sgemm (C++ function), 35
spla_zgemm (C++ function), 36
SplaContext (C++ type), 25
SplaDistributionType (C++ enum), 3
SplaDistributionType::SPLA_DIST_BLACS_BLOCK_CYCLIC (C++ enumerator), 3
SplaDistributionType::SPLA_DIST_MIRROR (C++ enumerator), 3
SplaError (C++ enum), 45
SplaError::SPLA_GPU_ALLOCATION_ERROR (C++ enumerator), 46

```

SplaError::SPLA\_GPU\_BLAS\_ERROR (C++ *enumera-*  
*tor*), 46  
 SplaError::SPLA\_GPU\_ERROR (C++ *enumerator*), 45  
 SplaError::SPLA\_GPU\_INVALID\_DEVICE\_POINTER\_ERROR  
 (C++ *enumerator*), 46  
 SplaError::SPLA\_GPU\_INVALID\_VALUE\_ERROR  
 (C++ *enumerator*), 46  
 SplaError::SPLA\_GPU\_LAUNCH\_ERROR (C++ *enumera-*  
*tor*), 46  
 SplaError::SPLA\_GPU\_NO\_DEVICE\_ERROR (C++ *enu-*  
*merator*), 46  
 SplaError::SPLA\_GPU\_SUPPORT\_ERROR (C++ *enu-*  
*merator*), 46  
 SplaError::SPLA\_INTERNAL\_ERROR (C++ *enumera-*  
*tor*), 45  
 SplaError::SPLA\_INVALID\_ALLOCATOR\_FUNCTION  
 (C++ *enumerator*), 46  
 SplaError::SPLA\_INVALID\_HANDLE\_ERROR (C++  
*enumerator*), 45  
 SplaError::SPLA\_INVALID\_PARAMETER\_ERROR  
 (C++ *enumerator*), 45  
 SplaError::SPLA\_INVALID\_POINTER\_ERROR (C++  
*enumerator*), 45  
 SplaError::SPLA\_MPI\_ALLOCATION\_ERROR (C++  
*enumerator*), 45  
 SplaError::SPLA\_MPI\_ERROR (C++ *enumerator*), 45  
 SplaError::SPLA\_MPI\_THREAD\_SUPPORT\_ERROR  
 (C++ *enumerator*), 45  
 SplaError::SPLA\_SUCCESS (C++ *enumerator*), 45  
 SplaError::SPLA\_UNKNOWN\_ERROR (C++ *enumera-*  
*tor*), 45  
 SplaFillMode (C++ *enum*), 3  
 SplaFillMode::SPLA\_FILL\_MODE\_FULL (C++ *enu-*  
*merator*), 3  
 SplaFillMode::SPLA\_FILL\_MODE\_LOWER (C++ *enu-*  
*merator*), 4  
 SplaFillMode::SPLA\_FILL\_MODE\_UPPER (C++ *enu-*  
*merator*), 4  
 SplaMatrixDistribution (C++ *type*), 31  
 SplaOperation (C++ *enum*), 3  
 SplaOperation::SPLA\_OP\_CONJ\_TRANSPOSE (C++  
*enumerator*), 3  
 SplaOperation::SPLA\_OP\_NONE (C++ *enumerator*), 3  
 SplaOperation::SPLA\_OP\_TRANSPOSE (C++ *enumera-*  
*tor*), 3  
 SplaProcessingUnit (C++ *enum*), 3  
 SplaProcessingUnit::SPLA\_PU\_GPU (C++ *enumera-*  
*tor*), 3  
 SplaProcessingUnit::SPLA\_PU\_HOST (C++ *enumera-*  
*tor*), 3