SPLA Documentation

Release 0.1.0

ETH Zurich, Simon Frasch

Apr 25, 2023

C++ API REFERENCE:

SPLA - Specialized Parallel Linear Algebra

CHAPTER

ONE

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.

CHAPTER

TWO

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 estimatex 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 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 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 estimatex 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.

CHAPTER

THREE

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 coloumns 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

Coloumn 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 coloumn block size used for matrix partitioning.

Parameters

colBlockSize – **[in]** Coloumn 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 coloumn major ordering of MPI ranks.

Parameters

- **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.

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 coloumn major process grid.
- **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.

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.

CHAPTER

FOUR

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.

CHAPTER

FIVE

GEMM - SSB

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

```
------ T     ------
|    |       |    |
|    |       |    |
------       ------
|    |       |    |          -------        -------
|    |       |    |          |  |  |        |  |  |
------   *   ------    +     -------  -->   -------
|    |       |    |          |  |  |        |  |  |
|    |       |    |          -------        -------
------       ------             C              C
|    |       |    |
|    |       |    |
------       ------
  A            B
```

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 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.

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.

CHAPTER

SIX

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.

```
  ------ T      ------
  |    |        |    |
  |    |        |    |
  ------        ------
  |    |        |    |        -------      -------
  |    |        |    |        |  |  |      |  |  |
  ------   *    ------    +   -------  --> -------
  |    |        |    |           |  |         |  |
  |    |        |    |           ----         ----
  ------        ------            C            C
  |    |        |    |
  |    |        |    |
  ------        ------
    A             B
```

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 refenrence 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 submatrix 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.

CHAPTER SEVEN

GEMM - SBS

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

```
------                           ------      ------
|    |                           |    |      |    |
|    |                           |    |      |    |
------                           ------      ------
|    |             -------       |    |      |    |
|    |             |  |  |       |    |      |    |
------       *     -------   +   ------ -->  ------
|    |             |  |  |       |    |      |    |
|    |             -------       |    |      |    |
------                B          ------      ------
|    |                           |    |      |    |
|    |                           |    |      |    |
------                           ------      ------
  A                                C            C
```

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 submatrix 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 ldC $\geq$ 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.

CHAPTER

EIGHT

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

CHAPTER NINE

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]** Procesing 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 compuations 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 estimatex 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 estimatex 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.

CHAPTER

TEN

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 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_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 coloumns 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.

CHAPTER

ELEVEN

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.

CHAPTER TWELVE

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.

```
------ T     ------
|    |       |    |
|    |       |    |
------       ------
|    |       |    |         -------          -------
|    |       |    |         |  |  |          |  |  |
------   *   ------     +   -------    -->   -------
|    |       |    |         |  |  |          |  |  |
|    |       |    |         -------          -------
------       ------            C                C
|    |       |    |
|    |       |    |
------       ------
  A            B
```

```
------ T     ------
|    |       |    |
|    |       |    |
------       ------
|    |       |    |         -------          -------
|    |       |    |         |  |  |          |  |  |
------   *   ------     +   -------    -->   -------
|    |       |    |            |  |          |  |  |
|    |       |    |            ----             ----
------       ------            C                C
|    |       |    |
|    |       |    |
------       ------
  A            B
```

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.

CHAPTER

THIRTEEN

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 refenrence 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 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.

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.

CHAPTER

FOURTEEN

GEMM - SBS

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

```
------                             ------          ------
|    |                             |    |          |    |
|    |                             |    |          |    |
------                             ------          ------
|    |            -------          |    |          |    |
|    |            |  |  |          |    |          |    |
------     *      -------     +    ------    -->   ------
|    |            |  |  |          |    |          |    |
|    |            -------          |    |          |    |
------               B             ------          ------
|    |                             |    |          |    |
|    |                             |    |          |    |
------                             ------          ------
  A                                  C               C
```

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 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 submatrix 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 ldC $\geq$ mLocal.
- **ctx** – **[in]** *Context*, which provides configuration settings and reusable resources.

SplaError **spla_pdgemm_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_pcgemm_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_pzgemm_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.

CHAPTER FIFTEEN

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 suppport 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.

INDEX