曦云® 系列通用计算 GPU

mcSPARSE API 参考

CSRD-23019-020-F3_V04　　2024-03-15

沐曦专有和三级保密信息　　本文档受 NDA 管控

声明

更新记录

版本	日期	更新说明
V04	2024-03-15	新增曦云®系列 GPU 产品信息
V03	2024-01-25	描述性修改及格式修正
V02	2023-12-29	描述性修改及格式修正
V01	2023-10-16	正式版本首次发布

目录

1 介绍

mcSPARSE 库是基于 MetaX MXMACA® 运行时（MXMACA 工具包的一部分）实现的。它提供了稀疏矩阵和向量的基本线性代数子程序 (BLAS)。该库旨在从 C++ 代码中调用。它使用 GPU 设备上运行的 MetaX 的 MXMACA 运行库。mcSPARSE 库的目标是处理具有大量 (结构性) 零元素的矩阵，这些零元素约占总条目的 95%。

要使用 mcSPARSE API，应用程序必须在 GPU 内存空间中分配所需的矩阵和矢量，将数据填入其中，调用所需的 mcSPARSE API 函数序列，然后将结果从 GPU 内存空间上传回主机。

mcSPARSE 的功能被划分为以下几个类别：

- **mcSPARSE 辅助函数**：描述可用于后续库调用的辅助函数。
- **mcSPARSE 转换函数**：描述对稀疏格式的矩阵执行操作以获得不同的矩阵格式。
- **mcSPARSE BLAS 1 级函数**：描述稀疏向量和稠密向量之间的操作。
- **mcSPARSE BLAS 2 级函数**：描述稀疏矩阵和稠密向量之间的操作。
- **mcSPARSE BLAS 3 级函数**：描述稀疏矩阵与稀疏或稠密矩阵之间的操作。
- **mcSPARSE 预处理函数**：描述稀疏预处理函数。
- **mcSPARSE 重排序函数**：描述对稀疏格式的矩阵执行重排序操作。
- **mcSPARSE 通用函数**：描述处理稀疏矩阵的通用操作。

1.1 命名规则

mcSPARSE 库函数可用于数据类型 `float`，`double`，`mcComplex` 和 `mcDoubleComplex`。稀疏 1 级、2 级和 3 级函数遵循以下命名规则：`mcsparse<t>[<matrix data format>]<operation>[<output matrix data format>]`。

其中，`<t>` 可以是 `S`，`D`，`C`，`Z` 或 `X`，分别对应数据类型 `float`，`double`，`mcComplex`，`mcDoubleComplex` 和通用类型。

`<matrix data format>` 可以是 `dense`，`coo`，`csr` 或 `csc`，分别对应稠密格式，坐标格式，压缩稀疏行格式和压缩稀疏列格式。

最后，`<operation>` 可以是 `axpyi`、`gthr`、`gthrz`、`roti` 或 `sctr`，对应 1 级函数；也可以是 `mv` 或 `sv`，对应 2 级函数；还可以是 `mm` 或 `sm`，对应 3 级函数。

所有函数的返回类型均为 `mcsparseStatus_t`，在接下来的章节中将详细解释。

2 使用 mcSPARSE API

本章介绍如何使用 mcSPARSE 库的 API。

2.1 安装 mcSPARSE

mcSPARSE 是与 MXMACA 工具包一同发布并安装的库。MXMACA 工具包的安装，参见《曦云系列通用计算 GPU 快速上手指南》。在安装 MXMACA 工具包后，请确保已设置环境变量 MACA_PATH。

```
export MACA_PATH=/opt/maca
```

mcSPARSE API 相关文件如下所示。

```
#header location:
${MACA_PATH}/include/mcsparse

#lib location:
${MACA_PATH}/lib/libmcsparse.so
```

在使用 mcSPARSE 库编译代码之前，请确保已设置环境变量 ISU_FASTMODEL 为 1。

```
export ISU_FASTMODEL=1
```

2.2 标量参数

在 mcSPARSE API 中，标量参数 alpha 和 beta 可以通过主机或设备上的引用来传递。返回标量结果的少数函数，比如 `nnz()`，会通过主机或设备上的引用返回结果值。虽然这些函数会立即返回，类似于那些返回矩阵和向量结果的函数，但是直到在 GPU 上执行完整个程序后，标量结果才会准备好。当用户尝试从主机读取这个标量结果时，需要适当地同步这个操作。

这一特性使得 mcSPARSE 库函数可以完全异步地使用流进行执行，即使 alpha 和 beta 是由前一个内核生成的。例如，当使用该库来实现线性系统和特征值问题的迭代求解方法时，就会出现这种情况。

3 mcSPARSE 索引和数据格式

mcSPARSE 库支持稠密向量、稀疏向量以及稠密矩阵和稀疏矩阵格式。

3.1 基准索引格式

该库支持以 0 为基数和以 1 为基数的两种索引方式。通过 `mcsparseIndexBase_t` 类型来选择基准索引，可作为独立参数或矩阵描述符 `mcsparseMatDescr_t` 类型中的字段进行传递。

3.1.1 向量格式

本节描述了稠密向量和稀疏向量的格式。

3.1.1.1 稠密格式

稠密向量用一个线性数据数组表示，线性地存储在内存中。

3.1.1.2 稀疏格式

稀疏向量使用两个数组来表示。

- 数据数组包含了等价的稠密格式中的非零值。
- 整数索引数组包含了等价的稠密格式中对应非零值的位置。

3.2 矩阵格式

在这节中讨论了矩阵的稠密格式和几种稀疏格式。

3.2.1 稠密格式

假设稠密矩阵 `x` 以列主序格式存储在内存中，并由以下参数表示。

m	(integer)	矩阵的行数。
n	(integer)	矩阵的列数。
ldX	(integer)	X 的主维度，必须大于或等于 m。如果 ldX 大于 m，那么 X 表示存储在内存中的较大矩阵的子矩阵。
X	(pointer)	指向包含矩阵元素的数据数组的指针。假定为 X 分配了足够的存储空间以容纳所有矩阵元素，并且 mcSPARSE 库函数可以访问子矩阵之外的值，但永远不会覆盖它们。

3.2.2 坐标格式 (COO)

m×n 稀疏矩阵 A 由以下参数以 COO 格式表示。

nnz	(integer)	该矩阵中非零元素的数量。
cooValA	(pointer)	指向长度为 nnz 的数据数组，该数组按行主序格式存储了矩阵 A 中所有非零值。
cooRowIndA	(pointer)	指向长度为 nnz 的整数数组，其中包含了与数组 cooValA 中对应元素的行索引。
cooColIndA	(pointer)	指向长度为 nnz 的整数数组，其中包含了与数组 cooValA 中对应元素的列索引。

假定 COO 格式的稀疏矩阵以行主格式存储。每个 COO 元素由行列对组成。COO 格式假定按行排序。支持已排序和未排序的列索引。

3.2.3 压缩稀疏行格式 (CSR)

CSR 格式与 COO 格式之间唯一的区别在于，CSR 格式中包含行索引的数组是压缩的。m×n 稀疏矩阵 A 由以下参数以 CSR 格式表示。

nnz	(integer)	矩阵中非零元素的数量。
csrValA	(pointer)	指向长度为 nnz 的数据数组，存储了矩阵 A 中所有非零值，采用行主序格式存储。
csrRowPtrA	(pointer)	指向长度为 m+1 的整数数组，存储了对数组 csrColIndA 和 csrValA 的索引。该数组的前 m 个元素包含第 i 行中第一个非零元素的索引 (其中，i=i,...,m) 而最后一个元素包含 nnz+csrRowPtrA(0) 一般而言，对于以零为基和以一为基的索引方式，csrRowPtrA(0) 分别为 0 和 1。
csrColIndA	(pointer)	指向长度为 nnz 的整数数组，包含数组 csrValA 中相应元素的列索引。

稀疏矩阵以 CSR 格式存储时，假定其采用行优先的 CSR 格式存储。对于稀疏矩阵，支持已排序和未排序的列索引。

3.2.4 压缩稀疏列格式 (CSC)

CSC 格式与 COO 格式有两个不同之处：在 CSC 格式中，矩阵以列主序存储并且对包含列索引的数组进行了压缩。一个大小为 m×n 的矩阵 A 由以下参数以 CSC 格式表示。

nnz	(integer)	该矩阵中非零元素的数量。
cscValA	(pointer)	指向长度为 nnz 数据数组的指针，该数组按列主序的格式存储了矩阵 A 所有非零值。
cscRowIndA	(pointer)	指向长度为 nnz 整数数组的指针，该数组包含了 cscValA 数组中对应元素的行索引。
cscColPtrA	(pointer)	指向长度为 n+1 整数数组的指针，该数组保存着数组 cscRowIndA 和 cscValA 的索引。该数组的前 n 个元素包含了第 i 行中第一个非零元素的索引 (对于 i=i,⋯,n)，而最后一个元素包含了 nnz+cscColPtrA(0)。通常，对于以零为基和以一为基的索引方式，cscColPtrA(0) 分别为 0 和 1。

注解： 以 CSC 格式表示的矩阵 A 与其以 CSC 格式表示的转置具有完全相同的内存布局 (反之亦然)。

3.2.5 块压缩稀疏行格式 (BSR)

CSR 格式和 BSR 格式之间唯一的区别是存储元素的格式。前者存储原始数据类型 (single、double、mcComplex 和 mcDoubleComplex)，而后者存储两维方块状的原始数据类型。BSR 格式有以下参数。

blockDim	(integer)	矩阵 A 的块维度。
mb	(integer)	矩阵 A 的行数。
nb	(integer)	矩阵 A 的块列数。
nnzb	(integer)	矩阵中非零块的数量。
bsrValA	(pointer)	指向保存矩阵 A 所有非零块元素数据数组的指针。这些块元素按列主序或行主序存储。
bsrRowPtrA	(pointer)	指向长度为 mb+1 整数数组的指针，该数组保存了 bsrColIndA 和 bsrValA 数组的索引。这个数组的前 mb 个元素包含了第 i 行非零块的索引对于 (i=1, ⋯, mb)，而最后一个元素包含了 nnzb+bsrRowPtrA(0)。通常，对于以零为基和以一为基的索引方式，cscColPtrA(0) 分别为 0 和 1。
bsrColIndA	(pointer)	指向长度为 nnzb 整数数组的指针，该数组包含了 bsrValA 数组中相应块对应的列索引。

和 CSR 格式一样，BSR 格式中的 (行，列) 索引按行主序存储。索引数组首先按行索引排序，然后在同一行内按列索引排序。

3.2.6 扩展块压缩稀疏行格式 (BSRX)

BSRX 格式与 BSR 格式相同，但数组 bsrRowPtrA 被拆分为两部分。每行的第一个非零块仍由数组 bsrRowPtrA 指定，这与 BSR 格式相同，但每行的最后一个非零块的位置由数组 bsrEndPtrA 指定。简而言之，BSRX 格式类似于 BSR 格式的 4 维向量变体。

矩阵 A 由以下参数以 BSRX 格式表示。

blockDim	(integer)	矩阵 A 的块维度
mb	(integer)	矩阵 A 的数。
nb	(integer)	矩阵 A 的块列数
nnzb	(integer)	矩阵中非零块的数量。
bsrValA	(pointer)	指向保存矩阵 A 所有非零块元素数据数组的针指。这些块元素按列主序或行主序存储。

下页继续

表 3.1 – 续上页

bsrRowPtrA	(pointer)	指向长度为 mb 整数数组的指针，该数组保存了 bsrColIndA 和 bsrValA 数组的索引。bsrRowPtrA(i) 是第 i 行块在 bsrColIndA 和 bsrValA 中的第一个非零块的位置。
bsrEndPtrA	(pointer)	指向长度为 mb 整数数组的指针，该数组保存了 bsrColIndA 和 bsrValA 数组的索引。bsrRowPtrA(i) 是第 i 行块在 bsrColIndA 和 bsrValA 中最后一个非零块的下一个位置。
bsrColIndA	(pointer)	指向长度为 nnzb 整数数组的指针，该数组包含 bsrValA 数组中相应块对应的列索引。

4 mcSPARSE 类型参考

4.1 数据类型

支持 `float`、`double`、`mcComplex` 和 `mcDoubleComplex` 数据类型。前两个是标准的 C 数据类型，而后两个是从 `mcComplex.h` 导出的。

4.2 mcsparseStatus_t

这种数据类型表示库函数返回的状态，其值如下：

值	含义
MCSPARSE_STATUS_SUCCESS	成功完成操作。
MCSPARSE_STATUS_NOT_INITIALIZED	mcSPARSE 库未初始化。这通常是由于缺少先前调用、mcSPARSE 例程调用 MXMACA Runtime API 发生错误，或者硬件设置错误。 **纠正方法：**在函数调用之前调用 `mcsparseCreate()`；并检查硬件、相应版本的驱动程序以及 mcSPARSE 库是否正确安装。 该错误也适用于通用 API，用于指示矩阵/向量描述符未初始化。
MCSPARSE_STATUS_ALLOC_FAILED	mcSPARSE 库内部资源分配失败。通常是由于设备内存分配 (`mcMalloc()`) 或主机内存分配失败。 **纠正方法：**在函数调用之前，尽可能地释放以前分配的内存。
MCSPARSE_STATUS_INVALID_VALUE	传递了一个无效值或参数 (例如负的向量大小) 给函数。 **纠正方法：**确保传递的所有参数都具有有效值。
MCSPARSE_STATUS_EXECUTION_FAILED	GPU 程序执行失败。这通常是由于 GPU 核启动失败，失败可能由多种原因导致。 **纠正方法：**检查硬件、相应版本的驱动程序及 mcSPARSE 库是否正确安装
MCSPARSE_STATUS_INTERNAL_ERROR	mcSPARSE 内部操作失败。 **纠正方法：**检查硬件、相应版本的驱动程序以及 mcSPARSE 库是否正确安装。同时检查是否在例程完成之前释放作为参数传递给例程的内存。
MCSPARSE_STATUS_MATRIX_TYPE_NOT_SUPPORTED	矩阵类型不受此函数支持。通常是由于向函数传递了无效的矩阵描述符。 **纠正方法：**检查 `mcsparseMatDescr_t descrA` 中的字段是否设置正确。
MCSPARSE_STATUS_NOT_SUPPORTED	该操作或数据类型组合不受此函数支持。
MCSPARSE_STATUS_INSUFFICIENT_RESOURCES	计算的资源。

4.3 mcsparseHandle_t

这是一个指向不透明 mcSPARSE 上下文的指针类型，用户必须在调用其他库函数之前通过调用 `mcsparseCreate()` 来初始化。`mcsparseCreate()` 创建并返回的句柄必须传递给每个 mcSPARSE 函数。

4.4 mcsparsePointerMode_t

此类型指示标量值是在主机上还是设备上通过引用传递。需要指出的是，如果在函数调用中通过引用传递了多个标量值，则所有标量值都将符合相同的单指针模式。指针模式可以分别使用 `mcsparseSetPointerMode()` 和 `mcsparseGetPointerMode()` 例程进行设置和检索。

值	含义
MCSPARSE_POINTER_MODE_HOST	标量值是在主机上通过引用传递。
MCSPARSE_POINTER_MODE_DEVICE	标量值是在设备上通过引用传递。

4.5 mcsparseOperation_t

此类型指示需要对稀疏矩阵执行哪些操作。

值	含义
MCSPARSE_OPERATION_NON_ TRANSPOSE	选择了非转置操作。
MCSPARSE_OPERATION_TRANSPOSE	选择了转置操作。
MCSPARSE_OPERATION_CONJUGATE_ TRANSPOSE	选择了共轭转置操作。

4.6 mcsparseAction_t

此类型指示操作是仅对索引进行，还是对数据和索引同时进行。

值	含义
MCSPARSE_ACTION_SYMBOLIC	操作仅在索引上执行。
MCSPARSE_ACTION_NUMERIC	操作同时在数据和索引上执行。

4.7 mcsparseDirection_t

此类型指示在函数 `mcsparse[S|D|C|Z]nnz` 中，稠密矩阵的元素是按行解析，还是按列解析（假设稠密矩阵按列主序存储在内存中）。此外，BSR 格式中块的存储格式也受此类型控制。

值	含义
MCSPARSE_DIRECTION_ROW	矩阵应该按行解析。
MCSPARSE_DIRECTION_COLUMN	矩阵应该按列解析。

4.8 mcsparseMatDescr_t

此结构体用于描述矩阵的形状和特性。

```
typedef struct {
      mcsparseMatrixType_t MatrixType;
      mcsparseFillMode_t FillMode;
      mcsparseDiagType_t DiagType;
      mcsparseIndexBase_t IndexBase;
} mcsparseMatDescr_t;
```

4.8.1 mcsparseDiagType_t

此类型指示矩阵的对角元素是否为 1。假定对角元素始终存在。如果将 MCSPARSE_DIAG_TYPE_UNIT 传递给 API 例程，则例程就会假定所有对角线元素都为 1，并且不会读取或修改这些元素。请注意，在这种情况下，例程假定对角线元素等于 1，而不管这些元素在内存中实际上被设置为何值。

值	含义
MCSPARSE_DIAG_TYPE_NON_UNIT	矩阵的对角线上具有非单位元素。
MCSPARSE_DIAG_TYPE_UNIT	矩阵的对角线上具有单位元素。

4.8.2 mcsparseFillMode_t

此类型指示矩阵的下部分还是上部分存储在稀疏存储中。

值	含义
MCSPARSE_FILL_MODE_LOWER	存储了矩阵的下三角部分。
MCSPARSE_FILL_MODE_UPPER	存储了矩阵的上三角部分。

4.8.3 mcsparseIndexBase_t

此类型指示矩阵索引的基数是 0 还是 1。

值	含义
MCSPARSE_INDEX_BASE_ZERO	基准索引是 0。
MCSPARSE_INDEX_BASE_ONE	基准索引是 1。

4.8.4 mcsparseMatrixType_t

此类型指示稀疏存储中所存储的矩阵类型。请注意，对于对称矩阵、厄米矩阵（Hermitian Matrix）和三角矩阵，只假定存储它们的下部分或上部分。

矩阵类型和填充模式的整体思想是为对称/厄米矩阵保持最小的存储，并利用 SpMV（稀疏矩阵向量乘）的对称属性。当 A 是对称矩阵且只存储下三角部分时，计算 y=A*x 需要两个步骤。第一步是计算 y=(L+D)*x，第二步是计算 y=L^T*x + y。考虑到转置操作 y=L^T*x 比非转置版本 y=L*x 慢 10 倍，对称特性并未表现出任何性能提升。对用户来说，最好将对称矩阵扩展为一般矩阵，并使用矩阵类型 MCSPARSE_MATRIX_TYPE_GENERAL 来计算 y=A*x。

一般来说，SpMV、预处理器（不完全 Cholesky 或不完全 LU）和三角求解器通常会在迭代求解器中结合使用，例如 PCG 和 GMRES。如果用户总是使用一般矩阵，而不是对称矩阵，则在预处理器中无需支持除

一般矩阵外的其他矩阵类型。因此，新的例程 [bsr|csr]sv2（三角求解器）、[bsr|csr]ilu02（不完全 LU）和 [bsr|csr]ic02（不完全 Cholesky）只支持矩阵类型 MCSPARSE_MATRIX_TYPE_GENERAL。

值	含义
MCSPARSE_MATRIX_TYPE_GENERAL	矩阵是一般矩阵。
MCSPARSE_MATRIX_TYPE_SYMMETRIC	矩阵是对称矩阵。
MCSPARSE_MATRIX_TYPE_HERMITIAN	矩阵是厄米矩阵。
MCSPARSE_MATRIX_TYPE_TRIANGULAR	矩阵是三角矩阵。

4.9 mcsparseAlgMode_t

用于表示 mcsparseCsrmvEx() 和 mcsparseCsrmvEx_bufferSize() 函数的算法参数类型。

值	含义
MCSPARSE_ALG_MERGE_PATH	使用适用于不规则非零模式的负载平衡算法

4.10 mcsparseColorInfo_t

指向不透明结构体的指针类型，该结构体包含 csrcolor() 中使用的信息。

4.11 mcsparseSolvePolicy_t

此类型指示在使用 csrsv2, csric02, csrilu02, bsrsv2, bsric02 和 bsrilu02 时是否生成和使用层级信息。

值	含义
MCSPARSE_SOLVE_POLICY_NO_LEVEL	没有生成和使用层级信息。
MCSPARSE_SOLVE_POLICY_USE_LEVEL	生成和使用层级信息。

4.12 mcsparseBsrsv2Info_t

指向不透明结构体的指针类型，该结构体包含在 bsrsv2_bufferSize()、bsrsv2_analysis() 和 bsrsv2_solve() 中使用的信息。

4.13 mcsparseCsrgemm2Info_t

指向不透明结构体的指针类型，该结构体包含在 csrgemm2_bufferSizeExt() 和 csrgemm2() 中使用的信息。

4.14 mcsparseCsric02Info_t

指向不透明结构体的指针类型，该结构体包含在 csric02_bufferSize()、csric02_analysis() 和 csric02() 中使用的信息。

4.15 mcsparseCsrilu02Info_t

指向不透明结构体的指针类型，该结构体包含在 csrilu02_bufferSize()、csrilu02_analysis() 和 csrilu02() 中使用的信息。

4.16 mcsparseCsrsm2Info_t

指向不透明结构体的指针类型，该结构体包含在 csrsm2_bufferSize()、csrsm2_analysis() 和 csrsm2_solve() 中使用的信息。

4.17 mcsparseCsrsv2Info_t

指向不透明结构体的指针类型，该结构体包含在 csrsv2_bufferSize()、csrsv2_analysis() 和 csrsv2_solve() 中使用的信息。

5 mcSPARSE 管理函数参考

本章节描述管理库的 mcSPARSE 函数。

5.1 mcsparseCreate()

```
mcsparseStatus_t
mcsparseCreate(mcsparseHandle_t *handle)
```

此函数初始化 mcSPARSE 库并在 mcSPARSE 上下文中创建一个句柄。必须在调用任何其他 mcSPARSE API 函数之前调用此函数。它分配访问 GPU 所需的硬件资源。

参数	输入/输出	含义
handle	输入	指向 mcSPARSE 上下文句柄的指针。

有关返回状态的描述，请参见mcsparseStatus_t。

5.2 mcsparseDestroy()

```
mcsparseStatus_t
mcsparseDestroy(mcsparseHandle_t handle)
```

此函数用于释放 mcSPARSE 库所使用的 CPU 端资源。GPU 端资源的释放可能会延迟到应用程序关闭时。

参数	输入/输出	含义
handle	输入	处理 mcSPARSE 上下文的句柄。

有关返回状态的描述，请参见mcsparseStatus_t。

5.3 mcsparseGetErrorName()

```
const char*
mcsparseGetErrorString(mcsparseStatus_t status)
```

此函数返回错误代码枚举名称的字符串表示。如果错误代码未被识别，则返回 `unrecognized error code`。

参数	输入/输出	含义
status	输入	要转换为字符串的错误代码。
const char*	输出	指向以 NULL 结尾字符串的指针。

5.4 mcsparseGetErrorString()

```
const char*
mcsparseGetErrorString(mcsparseStatus_t status)
```

返回错误代码的描述字符串。如果错误代码未被识别，则返回 unrecognized error code。

参数	输入/输出	含义
status	输入	要转换为字符串的错误代码。
const char*	输出	指向以 NULL 结尾字符串的指针。

5.5 mcsparseGetProperty()

```
mcsparseStatus_t
mcsparseGetProperty(libraryPropertyType type,
                    int*                value)
```

此函数返回请求属性的值。支持的属性类型，请参见下方 libraryPropertyType。

参数	输入/输出	含义
type	输入	请求的属性。
value	输出	请求属性的值。

libraryPropertyType （在 library_types.h 中定义）：

值	含义
MAJOR_VERSION	枚举器，用于查询主要版本号。
MINOR_VERSION	枚举器，用于查询次要版本号。
PATCH_LEVEL	用于标识补丁级别的编号。

有关返回状态的描述，请参见mcsparseStatus_t。

5.6 mcsparseGetVersion()

```
mcsparseStatus_t
mcsparseGetVersion(mcsparseHandle_t handle,
                   int*             version)
```

此函数返回 mcSPARSE 库的版本号。

参数	输入/输出	含义
handle	输入	mcSPARSE 句柄
version	输出	库的版本号

有关返回状态的描述，请参见mcsparseStatus_t。

5.7 mcsparseGetPointerMode()

```
mcsparseStatus_t
mcsparseGetPointerMode(mcsparseHandlet handle,
                       mcsparsePointerMode_t *mode)
```

此函数获取 mcSPARSE 库使用的指针模式。更多详细信息，请参见mcsparsePointerMode_t。

参数	输入/输出	含义
handle	输入	处理 mcSPARSE 上下文的句柄。
mode	输出	枚举指针模式类型之一。

有关返回状态的描述，请参见mcsparseStatus_t。

5.8 mcsparseSetPointerMode()

```
mcsparseStatus_t
mcsparseSetPointerMode(mcsparseHandle_t handle,
                       mcsparsePointerMode_t mode)
```

此函数设置 mcSPARSE 库使用的指针模式。默认是将值通过引用传递到主机上。更多详细信息，请参见mcsparsePointerMode_t。

参数	输入/输出	含义
handle	输入	处理 mcSPARSE 上下文的句柄。
mode	输入	枚举指针模式类型之一。

有关返回状态的描述，请参见mcsparseStatus_t。

5.9 mcsparseGetStream()

```
mcsparseStatus_t
mcsparseGetStream(mcsparseHandle_t handle, macaStream_t *streamId)
```

此函数获取 mcSPARSE 库的流，该流用于执行所有对 mcSPARSE 库函数的调用。如果 mcSPARSE 库的流未进行设置，则所有核都使用默认的 NULL 流。

参数	输入/输出	含义
handle	输入	处理 mcSPARSE 上下文的句柄。
streamId	输出	库使用的流 (stream)

有关返回状态的描述，请参见mcsparseStatus_t。

5.10 mcsparseSetStream()

```
mcsparseStatus_t
mcsparseSetStream(mcsparseHandle_t handle, macaStream_t streamId)
```

此函数设置 mcSPARSE 库要使用的流，用于执行其例程。

参数	输入/输出	含义
handle	输入	处理 mcSPARSE 上下文的句柄。
streamId	输入	库使用的流 (stream)

有关返回状态的描述，请参见mcsparseStatus_t。

6 mcSPARSE 辅助函数参考

6.1 mcsparseCreateColorInfo()

```
mcsparseStatus_t
mcsparseCreateColorInfo(mcsparseColorInfo_t* info)
```

此函数创建 mcsparseColorInfo_t 结构体，并初始化为默认值。

输入

info	指向 mcsparseColorInfo_t 结构体的指针

有关返回状态的描述，请参见mcsparseStatus_t 。

6.2 mcsparseCreateMatDescr()

```
mcsparseStatus_t
mcsparseCreateMatDescr(mcsparseMatDescr_t *descrA)
```

此函数初始化矩阵描述符。它将字段 MatrixType 和 IndexBase 分别设置为默认值 MCSPARSE_MATRIX_TYPE_GENERAL 和 MCSPARSE_INDEX_BASE_ZERO，其他字段保持未初始化。

输入

descrA	指向矩阵描述符的指针。

有关返回状态的描述，请参见mcsparseStatus_t。

6.3 mcsparseDestroyColorInfo()

```
mcsparseStatus_t
mcsparseDestroyColorInfo(mcsparseColorInfo_t info)
```

此函数销毁并释放结构体所需的内存。

输入

info	指向 csrcolor() 结构体的指针。

有关返回状态的描述，请参见mcsparseStatus_t。

6.4 mcsparseDestroyMatDescr()

```
mcsparseStatus_t
mcsparseDestroyMatDescr(mcsparseMatDescr_t descrA)
```

此函数释放分配给矩阵描述符的内存。

输入

descrA	矩阵描述符。

有关返回状态的描述，请参见mcsparseStatus_t。

6.5 mcsparseGetMatDiagType()

```
mcsparseDiagType_t
mcsparseGetMatDiagType(const mcsparseMatDescr_t descrA)
```

此函数返回矩阵描述符 descrA 的 DiagType 字段。

输入

descrA	矩阵描述符。

返回

DiagType	枚举 diagType 类型之一。

6.6 mcsparseGetMatFillMode()

```
mcsparseFillMode_t
mcsparseGetMatFillMode(const mcsparseMatDescr_t descrA)
```

此函数返回矩阵描述符 descrA 的 FillMode 字段。

输入

descrA	矩阵描述符。

返回

FillMode	枚举 fillMode 类型之一。

6.7 mcsparseGetMatIndexBase()

```
mcsparseIndexBase_t
mcsparseGetMatIndexBase(const mcsparseMatDescr_t descrA)
```

此函数返回矩阵描述符 descrA 的 IndexBase 字段

输入

descrA	矩阵描述符。

返回

IndexBase	枚举 indexBase 类型之一。

6.8 mcsparseGetMatType()

```
mcsparseMatrixType_t
mcsparseGetMatType(const mcsparseMatDescr_t descrA)
```

此函数返回矩阵描述符 descrA 的 MatrixType 字段。

输入

descrA	矩阵描述符。

返回

MatrixType	枚举 matrixType 类型之一。

6.9 mcsparseSetMatDiagType()

```
mcsparseStatus_t
mcsparseSetMatDiagType(mcsparseMatDescr_t descrA,
                       mcsparseDiagType_t diagType)
```

此函数设置矩阵描述符 descrA 的 DiagType 字段。

输入

diagType	枚举 diagType 类型之一。

输出

descrA	矩阵描述符。

有关返回状态的描述，请参见mcsparseStatus_t。

6.10 mcsparseSetMatFillMode()

```
mcsparseStatus_t
mcsparseSetMatFillMode(mcsparseMatDescr_t descrA,
                       mcsparseFillMode_t fillMode)
```

此函数设置矩阵描述符 descrA 的 FillMode 字段。

输入

fillMode	枚举 fillMode 类型之一。

输出

descrA	矩阵描述符。

有关返回状态的描述，请参见mcsparseStatus_t。

6.11 mcsparseSetMatIndexBase()

```
mcsparseStatus_t
mcsparseSetMatIndexBase(mcsparseMatDescr_t descrA,
                        mcsparseIndexBase_t base)
```

此函数设置矩阵描述符 descrA 的 IndexBase 字段。

输入

base	枚举 indexBase 类型之一。

输出

descrA	矩阵描述符。

有关返回状态的描述，请参见mcsparseStatus_t。

6.12 mcsparseSetMatType()

```
mcsparseStatus_t
mcsparseSetMatType(mcsparseMatDescr_t descrA, mcsparseMatrixType_t type)
```

此函数设置矩阵描述符 descrA 的 MatrixType 字段

输入

type	枚举 matrixType 类型之一。

输出

descrA	矩阵描述符。

有关返回状态的描述，请参见mcsparseStatus_t。

6.13 mcsparseCreateCsrsv2Info()

```
mcsparseStatus_t
mcsparseCreateCsrsv2Info(csrsv2Info_t *info);
```

此函数创建 csrsv2 的求解和分析结构，并初始化为默认值。

输入

info	指向 csrsv2 的求解和分析结构的指针。

有关返回状态的描述，请参见mcsparseStatus_t。

6.14 mcsparseDestroyCsrsv2Info()

```
mcsparseStatus_t
mcsparseDestroyCsrsv2Info(csrsv2Info_t info);
```

此函数销毁并释放结构体所需的内存。

输入

info	求解结构体 (csrsv2_solve) 和分析结构体 (csrsv2_analysis)

有关返回状态的描述，请参见mcsparseStatus_t。

6.15 mcsparseCreateCsrsm2Info()

```
mcsparseStatus_t
mcsparseCreateCsrsm2Info(csrsm2Info_t *info);
```

此函数创建 csrsm2 的求解和分析结构，并初始化为默认值。

输入

info	指向 csrsm2 的求解和分析结构的指针。

有关返回状态的描述，请参见mcsparseStatus_t。

6.16 mcsparseDestroyCsrsm2Info()

```
mcsparseStatus_t
mcsparseDestroyCsrsm2Info(csrsm2Info_t info);
```

此函数销毁并释放结构所需的内存。

输入

info	求解结构体 (csrsm2_solve) 和分析结构体 (csrsm2_analysis)。

有关返回状态的描述，请参见mcsparseStatus_t。

6.17 mcsparseCreateCsric02Info()

```
mcsparseStatus_t
mcsparseCreateCsric02Info(csric02Info_t *info);
```

此函数创建不完全 Cholesky 的求解结构体和分析结构体，并初始化为默认值。

输入

info	指向不完全 Cholesky 求解结构体和分析结构体的指针。

有关返回状态的描述，请参见mcsparseStatus_t。

6.18 mcsparseDestroyCsric02Info()

```
mcsparseStatus_t
mcsparseDestroyCsric02Info(csric02Info_t info);
```

此函数会销毁并释放由结构体所需的内存。

输入

info	求解结构体 (csric02_solve) 和分析结构体 (csric02_analysis)。

有关返回状态的描述，请参见mcsparseStatus_t。

6.19 mcsparseCreateCsrilu02Info()

```
mcsparseStatus_t
mcsparseCreateCsrilu02Info(csrilu02Info_t *info);
```

此函数创建不完全 LU 分解中的求解结构体和分析结构体，并初始化为默认值。

输入

info	指向不完全 LU 求解结构体和分析结构体的指针。

有关返回状态的描述，请参见mcsparseStatus_t。

6.20 mcsparseDestroyCsrilu02Info()

```
mcsparseStatus_t
mcsparseDestroyCsrilu02Info(csrilu02Info_t info);
```

此函数销毁并释放结构体所需的内存。

输入

info	求解结构体 (csrilu02_solve) 和分析结构体 (csrilu02_analysis)。

有关返回状态的描述，请参见mcsparseStatus_t。

6.21 mcsparseCreateBsrsv2Info()

```
mcsparseStatus_t
mcsparseCreateBsrsv2Info(bsrsv2Info_t *info);
```

此函数创建 BSRSV2 中的求解结构体和分析结构体，并初始化默认值。

输入

info	指向 BSRSV2 求解结构体和分析结构体的指针。

有关返回状态的描述，请参见mcsparseStatus_t。

6.22 mcsparseDestroyBsrsv2Info()

```
mcsparseStatus_t
mcsparseDestroyBsrsv2Info(bsrsv2Info_t info);
```

此函数销毁并释放结构体所需的内存。

输入

info	求解结构体 (bsrsv2_solve) 和分析结构体 (bsrsv2_analysis)。

有关返回状态的描述，请参见mcsparseStatus_t。

6.23 mcsparseCreateCsrgemm2Info()

```
mcsparseStatus_t
mcsparseCreateCsrgemm2Info(csrgemm2Info_t *info);
```

此函数创建一般稀疏矩阵乘法的分析结构体，并将其初始化。

输入

info	指向一般稀疏矩阵乘法分析结构体的指针。

有关返回状态的描述，请参见mcsparseStatus_t。

6.24 mcsparseDestroyCsrgemm2Info()

```
mcsparseStatus_t
mcsparseDestroyCsrgemm2Info(csrgemm2Info_t info);
```

此函数销毁并释放结构体所需的内存。

输入

info	csrgemm2 的不透明结构体。

有关返回状态的描述，请参见mcsparseStatus_t。

6.25 mcsparseCreatePruneInfo()

```
mcsparseStatus_t
mcsparseCreatePruneInfo(pruneInfo_t *info);
```

此函数创建了名为 prune 的结构体，并初始化为默认值。

输入

info	指向 prune 结构体的指针。

有关返回状态的描述，请参见mcsparseStatus_t。

6.26 mcsparseDestroyPruneInfo()

```
mcsparseStatus_t
mcsparseDestroyPruneInfo(pruneInfo_t info);
```

此函数销毁并释放了结构体所需的任何内存。

输入

info	prune 结构体。

有关返回状态的描述，请参见mcsparseStatus_t。

7 mcSPARSE 1 级函数参考

本章描述了执行稠密向量和稀疏向量之间操作的稀疏线性代数函数。

7.1 mcsparse<t>axpyi()

```
usparseStatus_t
mcsparseSaxpyi(mcsparseHandle_t    handle,
               int                 nnz,
               const float*        alpha,
               const float*        xVal,
               const int*          xInd,
               float*              y,
               mcsparseIndexBase_t idxBase)

mcsparseStatus_t
mcsparseDaxpyi(mcsparseHandle_t    handle,
               int                 nnz,
               const double*       alpha,
               const double*       xVal,
               const int*          xInd,
               double*             y,
               mcsparseIndexBase_t idxBase)

mcsparseStatus_t
mcsparseCaxpyi(mcsparseHandle_t    handle,
               int                 nnz,
               const mcComplex*    alpha,
               const mcComplex*    xVal,
               const int*          xInd,
               mcComplex*          y,
               mcsparseIndexBase_t idxBase)

mcsparseStatus_t
mcsparseZaxpyi(mcsparseHandle_t       handle,
               int                    nnz,
               const mcDoubleComplex* alpha,
               const mcDoubleComplex* xVal,
               const int*             xInd,
               mcDoubleComplex*       y,
               mcsparseIndexBase_t    idxBase)
```

此函数将稀疏向量 x 乘以常数 alpha，然后将结果添加到稠密向量 y 中。该操作可以写成以下形式：

```
for i=0 to nnz-1
      y[xInd[i]-idxBase] = y[xInd[i]-idxBase] + alpha*xVal[i]
```

- 该例程不需要额外的存储空间。
- 该例程支持异步执行。

输入

handle	处理 mcSPARSE 库上下文的句柄。
nnz	向量 x 的元素数量。
alpha	用于乘法运算的 <type> 标量。
xVal	包含向量 x nnz 个非零值的 <type> 向量。
xInd	包含向量 x nnz 个非零值的索引的整数向量。
y	以稠密格式表示的 <type> 向量。
idxBase	MCSPARSE_INDEX_BASE_ZERO 或 MCSPARSE_INDEX_BASE_ONE。

输出

y	以稠密格式表示的更新后的 <type> 向量 (如果 nnz == 0，则保持不变)。

有关返回状态的描述，请参见mcsparseStatus_t。

7.2 mcsparse<t>gthr()

```
mcsparseStatus_t
mcsparseSgthr(mcsparseHandle_t    handle,
              int                 nnz,
              const float*        y,
              float*              xVal,
              const int*          xInd,
              mcsparseIndexBase_t idxBase)

mcsparseStatus_t
mcsparseDgthr(mcsparseHandle_t    handle,
              int                 nnz,
              const double*       y,
              double*             xVal,
              const int*          xInd,
              mcsparseIndexBase_t idxBase)

mcsparseStatus_t
mcsparseCgthr(mcsparseHandle_t    handle,
              int                 nnz,
              const mcComplex*    y,
              mcComplex*          xVal,
              const int*          xInd,
              mcsparseIndexBase_t idxBase)

mcsparseStatus_t
mcsparseZgthr(mcsparseHandle_t        handle,
              int                     nnz,
              const mcDoubleComplex*  y,
              mcDoubleComplex*        xVal,
```

(下页继续)

(续上页)

```
                const int*              xInd,
                mcsparseIndexBase_t     idxBase)
```

此函数将向量 y 在索引数组 xInd 中列出的元素存储到数据数组 xVal 中。

- 该例程不需要额外的存储空间。
- 该例程支持异步执行。

输入

handle	处理 mcSPARSE 库上下文的句柄。
nnz	向量 x 的元素数量。
y	以稠密格式表示的 <type> 向量 (size>=max(xInd)-idxBase+1)。
xInd	包含向量 x nnz 个非零值的索引的整数向量。
idxBase	MCSPARSE_INDEX_BASE_ZERO 或 MCSPARSE_INDEX_BASE_ONE。

输出

xVal	<type> 向量，包含向量 y 向其中存储的 nnz 个非零值 (如果 nnz == 0，则保持不变)。

有关返回状态的描述，请参见mcsparseStatus_t。

7.3 mcsparse<t>gthrz()

```
mcsparseStatus_t
mcsparseSgthrz(mcsparseHandle_t    handle,
               int                 nnz,
               float*              y,
               float*              xVal,
               const int*          xInd,
               mcsparseIndexBase_t idxBase)

mcsparseStatus_t
mcsparseDgthrz(mcsparseHandle_t    handle,
               int                 nnz,
               double*             y,
               double*             xVal,
               const int*          xInd,
               mcsparseIndexBase_t idxBase)

mcsparseStatus_t
mcsparseCgthrz(mcsparseHandle_t    handle,
               int                 nnz,
               mcComplex*          y,
               mcComplex*          xVal,
               const int*          xInd,
               mcsparseIndexBase_t idxBase)

mcsparseStatus_t
mcsparseZgthrz(mcsparseHandle_t    handle,
               int                 nnz,
               mcDoubleComplex*    y,
               mcDoubleComplex*    xVal,
```

(下页继续)

(续上页)

```
                const int*           xInd,
                mcsparseIndexBase_t idxBase)
```

此函数将向量 y 在索引数组 xInd 中列出的元素存储到数据数组 xVal 中。同时将向量 y 中已转存的元素清零。

- 该例程不需要额外的存储空间。
- 该例程支持异步执行。

输入

handle	处理 mcSPARSE 库上下文的句柄。
nnz	向量 x 的元素数量。
y	以稠密格式表示的 <type> 向量 (size>=max(xInd)-idxBase+1)。
xInd	包含向量 x nnz 个非零值的索引的整数向量。
idxBase	MCSPARSE_INDEX_BASE_ZERO 或 MCSPARSE_INDEX_BASE_ONE。

输出

xVal	<type> 向量，包含向量 y 向其中存储的 nnz 个非零值 (如果 nnz == 0，则保持不变)。
y	以稠密格式表示的 <type> 向量，其中由 xInd 索引的元素设为零 (如果 nnz == 0，则保持不变)。

有关返回状态的描述，请参见mcsparseStatus_t。

7.4 mcsparse<t>roti()

```
mcsparseStatus_t
mcsparseSroti(mcsparseHandle_t    handle,
              int                  nnz,
              float*               xVal,
              const int*           xInd,
              float*               y,
              const float*         c,
              const float*         s,
              mcsparseIndexBase_t idxBase)

mcsparseStatus_t
mcsparseDroti(mcsparseHandle_t    handle,
              int                  nnz,
              double*              xVal,
              const int*           xInd,
              double*              y,
              const double*        c,
              const double*        s,
              mcsparseIndexBase_t idxBase)
```

此函数应用了 Givens 旋转矩阵。

```
for i=0 to nnz-1
      y[xInd[i]-idxBase] = c * y[xInd[i]-idxBase] - s*xVal[i]
      x[i]               = c * xVal[i]            + s * y[xInd[i]-idxBase]
```

- 该例程不需要额外的存储空间。

- 该例程支持异步执行。

输入

handle	处理 mcSPARSE 库上下文的句柄。
nnz	向量 x 的元素数量。
xVal	包含向量 x nnz 个非零值的 <type> 向量。
xInd	包含向量 x nnz 个非零值的索引的整数向量。
y	以稠密格式表示的 <type> 向量。
c	旋转矩阵的余弦元素。
s	旋转矩阵的正弦元素。
idxBase	MCSPARSE_INDEX_BASE_ZERO 或 MCSPARSE_INDEX_BASE_ONE。

输出

xVal	以稀疏格式更新了向量 <type> (如果 nnz == 0 则保持不变)。
y	以稠密格式表示的更新后的 <type> 向量 (如果 nnz == 0，则保持不变)。

有关返回状态的描述，请参见mcsparseStatus_t。

7.5 mcsparse<t>sctr()

```
mcsparseStatus_t
mcsparseSsctr(mcsparseHandle_t    handle,
              int                 nnz,
              const float*        xVal,
              const int*          xInd,
              float*              y,
              mcsparseIndexBase_t idxBase)

mcsparseStatus_t
mcsparseDsctr(mcsparseHandle_t    handle,
              int                 nnz,
              const double*       xVal,
              const int*          xInd,
              double*             y,
              mcsparseIndexBase_t idxBase)

mcsparseStatus_t
mcsparseCsctr(mcsparseHandle_t    handle,
              int                 nnz,
              const mcComplex*    xVal,
              const int*          xInd,
              mcComplex*          y,
              mcsparseIndexBase_t idxBase)

mcsparseStatus_t
mcsparseZsctr(mcsparseHandle_t       handle,
              int                    nnz,
              const mcDoubleComplex* xVal,
              const int*             xInd,
              mcDoubleComplex*       y,
              mcsparseIndexBase_t    idxBase)
```

此函数将稀疏向量 x 中的元素散布到稠密向量 y 中。此函数只修改了数组 xInd 中列出的索引对应的 y 中的元素。

- 该例程不需要额外的存储空间。
- 该例程支持异步执行。

输入

handle	处理 mcSPARSE 库上下文的句柄。
nnz	向量 x 的元素数量。
xVal	包含向量 x nnz 个非零值的 <type> 向量。
xInd	包含向量 x nnz 个非零值的索引的整数向量。
y	稠密向量 <type>(并且 size>=max(xInd)-idxBase+1)。
idxBase	MCSPARSE_INDEX_BASE_ZERO 或 MCSPARSE_INDEX_BASE_ONE。

输出

y	由向量 x 中的 nnz 个非零值散布得到的新向量 (如果 nnz == 0，则向量 x 保持不变)。

有关返回状态的描述，请参见mcsparseStatus_t。

8 mcSPARSE 2 级函数参考

这一章介绍了在稀疏矩阵和稠密向量之间执行相关操作的稀疏线性代数函数。

在求解稀疏三角线性系统的问题时，通常会分为两个阶段。首先，在分析阶段，通过调用适当的 `csrsv2_analysis()` 函数来分析稀疏三角矩阵，以确定其元素之间的依赖关系。该分析过程针对给定矩阵的稀疏模式和特定的 `mcsparseOperation_t` 函数类型。分析阶段的信息会存储在提前使用 `mcsparseCreateCsrsv2Info()` 函数初始化的 `csrsv2Info_t` 类型的参数中。

其次，在求解阶段，通过调用适当的 `csrsv2Info_t` 函数，使用存储在 `csrsv2Info_t` 参数中的信息来求解给定的稀疏三角线性系统。可以每次使用不同的右侧表达式来多次执行求解阶段的操作，但是分析阶段只能执行一次。当需要逐个求解一组系数矩阵保持不变但是拥有不同右侧表达式的稀疏三角线性系统时，这钟方法是非常有用的。

最后，一旦所有求解操作完成，`csrsv2Info_t` 参数指向的不透明数据结构将通过调用 `mcsparseDestroyCsrsv2Info()` 函数释放。

8.1 mcsparse<t>bsrmv()

```
mcsparseStatus_t
mcsparseSbsrmv(mcsparseHandle_t          handle,
               mcsparseDirection_t       dir,
               mcsparseOperation_t       trans,
               int                       mb,
               int                       nb,
               int                       nnzb,
               const float*              alpha,
               const mcsparseMatDescr_t  descr,
               const float*              bsrVal,
               const int*                bsrRowPtr,
               const int*                bsrColInd,
               int                       blockDim,
               const float*              x,
               const float*              beta,
               float*                    y)

mcsparseStatus_t
mcsparseDbsrmv(mcsparseHandle_t          handle,
               mcsparseDirection_t       dir,
               mcsparseOperation_t       trans,
               int                       mb,
               int                       nb,
               int                       nnzb,
               const double*             alpha,
               const mcsparseMatDescr_t  descr,
               const double*             bsrVal,
```

(下页继续)

(续上页)

```
                const int*                bsrRowPtr,
                const int*                bsrColInd,
                int                       blockDim,
                const double*             x,
                const double*             beta,
                double*                   y)

mcsparseStatus_t
mcsparseCbsrmv(mcsparseHandle_t           handle,
                mcsparseDirection_t       dir,
                mcsparseOperation_t       trans,
                int                       mb,
                int                       nb,
                int                       nnzb,
                const mcComplex*          alpha,
                const mcsparseMatDescr_t  descr,
                const mcComplex*          bsrVal,
                const int*                bsrRowPtr,
                const int*                bsrColInd,
                int                       blockDim,
                const mcComplex*          x,
                const mcComplex*          beta,
                mcComplex*                y)

mcsparseStatus_t
mcsparseZbsrmv(mcsparseHandle_t           handle,
                mcsparseDirection_t       dir,
                mcsparseOperation_t       trans,
                int                       mb,
                int                       nb,
                int                       nnzb,
                const mcDoubleComplex*    alpha,
                const mcsparseMatDescr_t  descr,
                const mcDoubleComplex*    bsrVal,
                const int*                bsrRowPtr,
                const int*                bsrColInd,
                int                       blockDim,
                const mcDoubleComplex*    x,
                const mcDoubleComplex*    beta,
                mcDoubleComplex*          y)
```

此函数执行矩阵-向量操作。

`bsrmv()` 具有以下特性：

- 该例程不需要额外的存储空间。
- 该例程支持异步执行。

`bsrmv()` 的几个注意事项：

- 仅支持 `blockDim > 1`
- 仅支持 `MCSPARSE_OPERATION_NON_TRANSPOSE`
- 仅支持 `MCSPARSE_MATRIX_TYPE_GENERAL`

例如，假设用户有一个 CSR 格式的矩阵，并希望尝试使用 `bsrmv()` 函数。下面的代码说明了如何在单精度计算中使用 `csr2bsr()` 转换和 `bsrmv()` 乘法。

```
// 假设 A 是一个用 CSR 格式表示的 m x n 的稀疏矩阵。
// hx 是一个大小为 n 的主机向量，hy 是一个大小为 m 的主机向量。
// m 和 n 不是 blockDim 的倍数。
// 步骤 1：将 CSR 转换为列优先的 BSR 格式
int base, nnz;
int nnzb;
mcsparseDirection_t dirA = MCSPARSE_DIRECTION_COLUMN;
int mb = (m + blockDim-1)/blockDim;
int nb = (n + blockDim-1)/blockDim;
mcMalloc((void**)&bsrRowPtrC, sizeof(int) *(mb+1));
mcsparseXcsr2bsrNnz(handle, dirA, m, n,
        descrA, csrRowPtrA, csrColIndA, blockDim,
        descrC, bsrRowPtrC, &nnzb);
mcMalloc((void**)&bsrColIndC, sizeof(int)*nnzb);
mcMalloc((void**)&bsrValC, sizeof(float)*(blockDim*blockDim)*nnzb);
mcsparseScsr2bsr(handle, dirA, m, n,
        descrA, csrValA, csrRowPtrA, csrColIndA, blockDim,
        descrC, bsrValC, bsrRowPtrC, bsrColIndC);
// 第二步，为 bsrmv 分配足够大的向量 x 和向量 y
mcMalloc((void**)&x, sizeof(float)*(nb*blockDim));
mcMalloc((void**)&y, sizeof(float)*(mb*blockDim));
mcMemcpy(x, hx, sizeof(float)*n, mcMemcpyHostToDevice);
mcMemcpy(y, hy, sizeof(float)*m, mcMemcpyHostToDevice);
// 第三步：执行 bsrmv
mcsparseSbsrmv(handle, dirA, transA, mb, nb, nnzb, &alpha,
   descrC, bsrValC, bsrRowPtrC, bsrColIndC, blockDim, x, &beta, y);
```

输入

handle	处理 mcSPARSE 库上下文的句柄。
dir	块的存储格式，可以是 MCSPARSE_DIRECTION_ROW 或 MCSPARSE_DIRECTION_COLUMN。
trans	仅支持 MCSPARSE_OPERATION_NON_TRANSPOSE
mb	矩阵的块行数
nb	矩阵的块列数。
nnzb	矩阵的非零块数。
alpha	用于乘法运算的 <type> 标量。
descr	矩阵的描述符。支持的矩阵类型有 MCSPARSE_MATRIX_TYPE_GENERAL 此外，支持的索引基准有 MCSPARSE_INDEX_BASE_ZERO 和 MCSPARSE_INDEX_BASE_ONE。
bsrVal	存储矩阵非零块的 <type> 数组。
bsrRowPtr	包含每个块行的起始位置和最后一个块行结束位置后移一位的整数数组，数组长度为 mb。
bsrColInd	存储了矩阵非零块的列索引的整数数组
blockDim	稀疏矩阵的块维度，大于零。
x	指向 <type> 元素的向量。
beta	<type> 标量用于乘法运算。如果 beta 为零，y 可以不是有效的输入。
y	指向 <type> 元素的向量。

输出

y	更新后的 <type> 向量。

有关返回状态的描述，请参见mcsparseStatus_t。

8.2 mcsparse<t>bsrxmv()

```
mcsparseStatus_t
mcsparseSbsrxmv(mcsparseHandle_t          handle,
                mcsparseDirection_t       dir,
                mcsparseOperation_t       trans,
                int                       sizeOfMask,
                int                       mb,
                int                       nb,
                int                       nnzb,
                const float*              alpha,
                const mcsparseMatDescr_t  descr,
                const float*              bsrVal,
                const int*                bsrMaskPtr,
                const int*                bsrRowPtr,
                const int*                bsrEndPtr,
                const int*                bsrColInd,
                int                       blockDim,
                const float*              x,
                const float*              beta,
                float*                    y)

mcsparseStatus_t
mcsparseDbsrxmv(mcsparseHandle_t          handle,
                mcsparseDirection_t       dir,
                mcsparseOperation_t       trans,
                int                       sizeOfMask,
                int                       mb,
                int                       nb,
                int                       nnzb,
                const double*             alpha,
                const mcsparseMatDescr_t  descr,
                const double*             bsrVal,
                const int*                bsrMaskPtr,
                const int*                bsrRowPtr,
                const int*                bsrEndPtr,
                const int*                bsrColInd,
                int                       blockDim,
                const double*             x,
                const double*             beta,
                double*                   y)

mcsparseStatus_t
mcsparseCbsrxmv(mcsparseHandle_t          handle,
                mcsparseDirection_t       dir,
                mcsparseOperation_t       trans,
                int                       sizeOfMask,
                int                       mb,
                int                       nb,
                int                       nnzb,
                const mcComplex*          alpha,
                const mcsparseMatDescr_t  descr,
                const mcComplex*          bsrVal,
                const int*                bsrMaskPtr,
                const int*                bsrRowPtr,
                const int*                bsrEndPtr,
```

(下页继续)

(续上页)

```
                  const int*                bsrColInd,
                  int                       blockDim,
                  const mcComplex*          x,
                  const mcComplex*          beta,
                  mcComplex*                y)

mcsparseStatus_t
mcsparseZbsrxmv(mcsparseHandle_t          handle,
                  mcsparseDirection_t       dir,
                  mcsparseOperation_t       trans,
                  int                       sizeOfMask,
                  int                       mb,
                  int                       nb,
                  int                       nnzb,
                  const mcDoubleComplex*    alpha,
                  const mcsparseMatDescr_t  descr,
                  const mcDoubleComplex*    bsrVal,
                  const int*                bsrMaskPtr,
                  const int*                bsrRowPtr,
                  const int*                bsrEndPtr,
                  const int*                bsrColInd,
                  int                       blockDim,
                  const mcDoubleComplex*    x,
                  const mcDoubleComplex*    beta,
                  mcDoubleComplex*          y)
```

此函数执行了一个 bsrmv 操作和一个掩码操作。

bsrmv() 具有以下特性：

- 该例程不需要额外的存储空间。
- 该例程支持异步执行。

bsrmv() 的几个注意事项：

- 仅支持 blockDim > 1。
- 仅支持 MCSPARSE_OPERATION_NON_TRANSPOSE 和 MCSPARSE_MATRIX_TYPE_GENERAL。
- 参数 bsrMaskPtr，bsrRowPtr，bsrEndPtr 与 bsrColInd 与基准索引一致，可以是基于一的基准索引或基于零的基准索引。上面的示例是基数为一的基准索引。

输入

handle	处理 mcSPARSE 库上下文的句柄。
dir	块的存储格式，可以是 MCSPARSE_DIRECTION_ROW 或 MCSPARSE_DIRECTION_COLUMN。
trans	仅支持 MCSPARSE_OPERATION_NON_TRANSPOSE
sizeOfMask	y 中被更新的块行的数量。
mb	矩阵的块行数。
nb	矩阵的块列数。
nnzb	矩阵的非零块数。
alpha	用于乘法运算的 <type> 标量。
descr	矩阵的描述符。支持的矩阵类型有 MCSPARSE_MATRIX_TYPE_GENERAL 此外，支持的索引基准有 MCSPARSE_INDEX_BASE_ZERO 和 MCSPARSE_INDEX_BASE_ONE。
bsrVal	矩阵的 nnz 个非零块的 <type> 数组。
bsrMaskPtr	存储与更新的块行相对应索引的长度为 sizeOfMask 整数数组。
bsrRowPtr	包含每个块行的起始位置的整数数组，数组长度为 mb。
bsrEndPtr	包含最后一个块行结束位置后移一位的整数数组，数组长度为 mb。
bsrColInd	存储了矩阵非零块 nnzb 个列索引信息的整数数组。
blockDim	稀疏矩阵的块维度，大于零。
x	指向元素的 <type> 向量。
beta	<type> 标量用于乘法运算。如果 beta 为零，y 可以不是有效的输入。
y	元素的 <type> 向量。

有关返回状态的描述，请参见mcsparseStatus_t。

8.3 mcsparse<t>bsrsv2_bufferSize()

```
mcsparseStatus_t
mcsparseSbsrsv2_bufferSize(mcsparseHandle_t          handle,
                           mcsparseDirection_t       dirA,
                           mcsparseOperation_t       transA,
                           int                       mb,
                           int                       nnzb,
                           const mcsparseMatDescr_t  descrA,
                           float*                    bsrValA,
                           const int*                bsrRowPtrA,
                           const int*                bsrColIndA,
                           int                       blockDim,
                           bsrsv2Info_t              info,
                           int*                      pBufferSizeInBytes)

mcsparseStatus_t
mcsparseDbsrsv2_bufferSize(mcsparseHandle_t          handle,
                           mcsparseDirection_t       dirA,
                           mcsparseOperation_t       transA,
                           int                       mb,
                           int                       nnzb,
                           const mcsparseMatDescr_t  descrA,
                           double*                   bsrValA,
                           const int*                bsrRowPtrA,
                           const int*                bsrColIndA,
                           int                       blockDim,
                           bsrsv2Info_t              info,
                           int*                      pBufferSizeInBytes)
```

(下页继续)

(续上页)

```
mcsparseStatus_t
mcsparseCbsrsv2_bufferSize(mcsparseHandle_t         handle,
                           mcsparseDirection_t      dirA,
                           mcsparseOperation_t      transA,
                           int                      mb,
                           int                      nnzb,
                           const mcsparseMatDescr_t descrA,
                           mcComplex*               bsrValA,
                           const int*               bsrRowPtrA,
                           const int*               bsrColIndA,
                           int                      blockDim,
                           bsrsv2Info_t             info,
                           int*                     pBufferSizeInBytes)

mcsparseStatus_t
mcsparseZbsrsv2_bufferSize(mcsparseHandle_t         handle,
                           mcsparseDirection_t      dirA,
                           mcsparseOperation_t      transA,
                           int                      mb,
                           int                      nnzb,
                           const mcsparseMatDescr_t descrA,
                           mcDoubleComplex*         bsrValA,
                           const int*               bsrRowPtrA,
                           const int*               bsrColIndA,
                           int                      blockDim,
                           bsrsv2Info_t             info,
                           int*                     pBufferSizeInBytes)
```

此函数返回在新稀疏三角线性系统 bsrsv2 中使用的缓冲区大小。虽然参数 trans 和矩阵 A 的上 (下) 三角部分有六种组合，但是 bsrsv2_bufferSize() 仅会返回这些组合中最大缓冲区的大小。缓冲区的大小取决于维度 mb，blockDim 以及矩阵的非零块数 nnzb。如果用户更改了矩阵，则必须再次调用 bsrsv2_bufferSize() 来获取正确的缓冲区大小；否则可能会导致段错误。

- 该例程不需要额外的存储空间。
- 该例程支持异步执行。

输入

handle	处理 mcSPARSE 库上下文的句柄。
dir	块的存储格式，可以是 MCSPARSE_DIRECTION_ROW 或 MCSPARSE_DIRECTION_COLUMN。
transA	op(A) 操作
mb	矩阵 A 的块行数。
nnzb	矩阵 A 中非零块的数量。
descrA	矩阵 A 的描述符。支持的矩阵类型有 MCSPARSE_MATRIX_TYPE_GENERAL 此外，支持的索引基准有 MCSPARSE_INDEX_BASE_ZERO 和 MCSPARSE_INDEX_BASE_ONE。
bsrValA	存储矩阵 A nnzb 个非零块的 <type> 数组。
bsrRowPtrA	包含每个块行的起始位置和最后一个块行结束位置后移一位的整数数组，数组长度为 mb。
bsrColIndA	存储了矩阵 A 中非零块 nnzb 个列索引信息的整数数组。
blockDim	稀疏矩阵 A 的块维度，必须大于零。

输出

info	根据不同的算法记录的内部状态。
pBufferSizeInBytes	在 bsrsv2_analysis() 和 bsrsv2_solve() 函数中使用的缓冲区的字节数。

有关返回状态的描述，请参见mcsparseStatus_t。

8.4 mcsparse<t>bsrsv2_analysis()

```
mcsparseStatus_t
mcsparseSbsrsv2_analysis(mcsparseHandle_t         handle,
                         mcsparseDirection_t      dirA,
                         mcsparseOperation_t      transA,
                         int                      mb,
                         int                      nnzb,
                         const mcsparseMatDescr_t descrA,
                         const float*             bsrValA,
                         const int*               bsrRowPtrA,
                         const int*               bsrColIndA,
                         int                      blockDim,
                         bsrsv2Info_t             info,
                         mcsparseSolvePolicy_t    policy,
                         void*                    pBuffer)

mcsparseStatus_t
mcsparseDbsrsv2_analysis(mcsparseHandle_t         handle,
                         mcsparseDirection_t      dirA,
                         mcsparseOperation_t      transA,
                         int                      mb,
                         int                      nnzb,
                         const mcsparseMatDescr_t descrA,
                         const double*            bsrValA,
                         const int*               bsrRowPtrA,
                         const int*               bsrColIndA,
                         int                      blockDim,
                         bsrsv2Info_t             info,
                         mcsparseSolvePolicy_t    policy,
                         void*                    pBuffer)

mcsparseStatus_t
mcsparseDbsrsv2_analysis(mcsparseHandle_t         handle,
                         mcsparseDirection_t      dirA,
                         mcsparseOperation_t      transA,
                         int                      mb,
                         int                      nnzb,
                         const mcsparseMatDescr_t descrA,
                         const mcComplex*         bsrValA,
                         const int*               bsrRowPtrA,
                         const int*               bsrColIndA,
                         int                      blockDim,
                         bsrsv2Info_t             info,
                         mcsparseSolvePolicy_t    policy,
                         void*                    pBuffer)

mcsparseStatus_t
```

(下页继续)

(续上页)

```
mcsparseZbsrsv2_analysis(mcsparseHandle_t          handle,
                         mcsparseDirection_t       dirA,
                         mcsparseOperation_t       transA,
                         int                       mb,
                         int                       nnzb,
                         const mcsparseMatDescr_t  descrA,
                         const mcDoubleComplex*    bsrValA,
                         const int*                bsrRowPtrA,
                         const int*                bsrColIndA,
                         int                       blockDim,
                         bsrsv2Info_t              info,
                         mcsparseSolvePolicy_t     policy,
                         void*                     pBuffer)
```

此函数执行新稀疏三角线性系统 bsrsv2 的分析阶段。BSR 格式的块大小为 blockDim*blockDim，根据参数 dirA 的设置可以按列主序或行主序存储。参数 dirA 可以为 MCSPARSE_DIRECTION_COLUMN 或 MCSPARSE_DIRECTION_ROW。矩阵类型必须是 MCSPARSE_MATRIX_TYPE_GENERAL，而填充模式和对角线类型则没有要求。

此函数预计只会在给定矩阵和特定操作类型下执行一次。

此函数需要由 bsrsv2_bufferSize() 返回缓冲区大小。pBuffer 的地址必须是 128 字节的倍数，如果不是，将返回 MCSPARSE_STATUS_INVALID_VALUE 参数。

函数 bsrsv2_analysis() 报告了存储在不透明结构 info 中的一个结构零与计算级别信息。级别信息可以提高三角求解器的并行效率。然而，bsrsv2_solve() 可以在没有级别信息的情况下完成求解。要禁用级别信息，用户需要将三角求解器的策略指定为 MCSPARSE_SOLVE_POLICY_NO_LEVEL。

函数 bsrsv2_analysis() 总是报告第一个结构零，即使参数 policy 为 MCSPARSE_SOLVE_POLICY_NO_LEVEL。指定为 MCSPARSE_DIAG_TYPE_UNIT 时，即使某些 j 的块 A(j,j) 缺失，也不会报告任何结构零。用户需要调用 mcsparseXbsrsv2_zeroPivot() 来确定结构零的位置。

如果 bsrsv2_analysis() 报告了一个结构零，用户可以选择是否调用 bsrsv2_solve()。在这种情况下，用户仍然可以调用 bsrsv2_solve()，此函数将在与结构零相同的位置返回一个数值零。然而，结果 x 是没有意义的。

- 此函数需要内部分配的临时额外存储空间。
- 如果流式有序内存分配器是可用的话，该例程将支持异步执行。

输入

handle	处理 mcSPARSE 库上下文的句柄。
dir	块的存储格式，可以是 MCSPARSE_DIRECTION_ROW 或 MCSPARSE_DIRECTION_COLUMN。
transA	op(A) 操作
mb	矩阵 A 的块行数。
nnzb	矩阵 A 中非零块的数量。
descrA	矩阵 A 的描述符。支持的矩阵类型有 MCSPARSE_MATRIX_TYPE_GENERAL 此外，支持的基准索引有 MCSPARSE_INDEX_BASE_ZERO 和 MCSPARSE_INDEX_BASE_ONE。
bsrValA	存储矩阵 A nnzb 个非零块的 <type> 数组。
bsrRowPtrA	包含每个块行的起始位置和最后一个块行结束位置后移一位的整数数组，数组长度为 mb。
bsrColIndA	存储了矩阵 A 中非零块 nnzb 个列索引信息的整数数组。
blockDim	稀疏矩阵 A 的块维度，大于零
info	mcsparseCreateBsrsv2Info() 初始化的结构。
policy	支持的策略有 MCSPARSE_SOLVE_POLICY_NO_LEVEL 和 MCSPARSE_SOLVE_POLICY_USE_LEVEL。
pBuffer	用户分配的缓冲区，其大小由 bsrsv2_bufferSize() 返回。

输出

info	将在分析阶段收集的信息填充到结构中 (应该在传递给求解阶段时保持不变)。

有关返回状态的描述，请参见mcsparseStatus_t。

8.5 mcsparse<t>bsrsv2_solve()

```
mcsparseStatus_t
mcsparseSbsrsv2_solve(mcsparseHandle_t         handle,
                      mcsparseDirection_t      dirA,
                      mcsparseOperation_t      transA,
                      int                      mb,
                      int                      nnzb,
                      const float*             alpha,
                      const mcsparseMatDescr_t descrA,
                      const float*             bsrValA,
                      const int*               bsrRowPtrA,
                      const int*               bsrColIndA,
                      int                      blockDim,
                      bsrsv2Info_t             info,
                      const float*             x,
                      float*                   y,
                      mcsparseSolvePolicy_t    policy,
                      void*                    pBuffer)

mcsparseStatus_t
mcsparseDbsrsv2_solve(mcsparseHandle_t         handle,
                      mcsparseDirection_t      dirA,
                      mcsparseOperation_t      transA,
                      int                      mb,
                      int                      nnzb,
                      const double*            alpha,
                      const mcsparseMatDescr_t descrA,
```

(下页继续)

(续上页)

```
                        const double*            bsrValA,
                        const int*               bsrRowPtrA,
                        const int*               bsrColIndA,
                        int                      blockDim,
                        bsrsv2Info_t             info,
                        const double*            x,
                        double*                  y,
                        mcsparseSolvePolicy_t    policy,
                        void*                    pBuffer)

mcsparseStatus_t
mcsparseCbsrsv2_solve(mcsparseHandle_t         handle,
                        mcsparseDirection_t      dirA,
                        mcsparseOperation_t      transA,
                        int                      mb,
                        int                      nnzb,
                        const mcComplex*         alpha,
                        const mcsparseMatDescr_t descrA,
                        const mcComplex*         bsrValA,
                        const int*               bsrRowPtrA,
                        const int*               bsrColIndA,
                        int                      blockDim,
                        bsrsv2Info_t             info,
                        const mcComplex*         x,
                        mcComplex*               y,
                        mcsparseSolvePolicy_t    policy,
                        void*                    pBuffer)

mcsparseStatus_t
mcsparseZbsrsv2_solve(mcsparseHandle_t         handle,
                        mcsparseDirection_t      dirA,
                        mcsparseOperation_t      transA,
                        int                      mb,
                        int                      nnzb,
                        const mcDoubleComplex*   alpha,
                        const mcsparseMatDescr_t descrA,
                        const mcDoubleComplex*   bsrValA,
                        const int*               bsrRowPtrA,
                        const int*               bsrColIndA,
                        int                      blockDim,
                        bsrsv2Info_t             info,
                        const mcDoubleComplex*   x,
                        mcDoubleComplex*         y,
                        mcsparseSolvePolicy_t    policy,
                        void*                    pBuffer)
```

此函数执行新稀疏三角线性系统 bsrsv2 的求解阶段。BSR 格式中的块大小为 blockDim*blockDim，并根据参数 dirA 确定是按列主序 (column-major) 还是按行主序 (row-major) 存储。dirA 参数的取值可以是 MCSPARSE_DIRECTION_COLUMN 或 MCSPARSE_DIRECTION_ROW。矩阵类型必须为 MCSPARSE_MATRIX_TYPE_GENERAL，而填充模式和对角线类型则没有要求。函数 bsrsv02_solve() 可以支持任意的 blockDim 大小。

此函数预计会在给定矩阵和特定操作类型下多次执行。

此函数需要由 bsrsv2_bufferSize() 返回缓冲区大小。pBuffer 的地址必须是 128 字节的倍数，如果不是，将返回 MCSPARSE_STATUS_INVALID_VALUE 参数。

尽管 bsrsv2_solve() 可以在没有级别信息的情况下执行，用户仍然需要注意一致性。如

果使用策略 MCSPARSE_SOLVE_POLICY_USE_LEVEL 来调用 bsrsv2_analysis()，则使不使用级别都能运行 bsrsv2_solve()。另外，如果使用 MCSPARSE_SOLVE_POLICY_NO_LEVEL 调用 bsrsv2_analysis()，那么 bsrsv2_solve() 只能接受 MCSPARSE_SOLVE_POLICY_NO_LEVEL 来运行；否则，将返回 MCSPARSE_STATUS_INVALID_VALUE。

级别信息可能不仅不会提高性能，还可能会花费额外的时间进行分析。例如，三对角矩阵没有并行性。在这种情况下，MCSPARSE_SOLVE_POLICY_NO_LEVEL 比 MCSPARSE_SOLVE_POLICY_USE_LEVEL 性能更好。如果用户有一个迭代求解器，最好的方法是使用 MCSPARSE_SOLVE_POLICY_USE_LEVEL 进行一次 bsrsv2_analysis()。然后，在第一次运行中使用 MCSPARSE_SOLVE_POLICY_NO_LEVEL 进行 bsrsv2_solve()，在第二次运行中使用 MCSPARSE_SOLVE_POLICY_USE_LEVEL，然后选择最快的方法来执行剩余的迭代。

函数 bsrsv02_solve() 的行为与 csrsv02_solve() 相同。也就是说，bsr2csr(bsrsv02(A)) = csrsv02(bsr2csr(A))。csrsv02_solve() 中的数值零意味着存在某个零元素 A(j,j)。bsrsv02_solve() 中的数值零意味着存在某个不可逆的块 A(j,j)。

函数 bsrsv2_solve() 会报告第一个数值零，包括结构零。指定 MCSPARSE_DIAG_TYPE_UNIT 时，即使 A(j,j) 对于某些 j 不可逆，也不会报告数值零。用户需要调用 mcsparseXbsrsv2_zeroPivot() 来了解数值零的位置。

如果 pBuffer != NULL，此函数将支持以下属性。

- 该例程不需要额外的存储空间。
- 该例程支持异步执行。

例如，假设 L 是一个具有单位对角线的下三角矩阵，那么以下代码可以通过级别信息求解有关 L*y=x 的问题。

```
// 假设 L 是由 BSR 格式表示的 m x m 稀疏矩阵。
// 块行/列的数量为 mb，
// 非零块的数量为 nnzb。
// L 是下三角矩阵，并且具有单位对角线。
// 假设：
// - 矩阵 L 的维度为 m(=mb*blockDim)，
// - 矩阵 L 有 nnz(=nnzb*blockDim*blockDim) 个非零元素，
// - 句柄已经通过 mcsparseCreate() 函数创建，
// - (d_bsrRowPtr, d_bsrColInd, d_bsrVal) 是存储在设备内存中的 L 的
BSR 格式，
// - d_x 是存储在设备内存中的右侧向量，
// - d_y 是存储在设备内存中的解向量，
// - d_x 和 d_y 的大小为 m。
mcsparseMatDescr_t descr = 0;
bsrsv2Info_t info = 0;
int pBufferSize;
void *pBuffer = 0;
int structural_zero;
int numerical_zero;
const double alpha = 1.;
const mcsparseSolvePolicy_t policy = MCSPARSE_SOLVE_POLICY_USE_
↪LEVEL;
const mcsparseOperation_t trans = MCSPARSE_OPERATION_NON_TRANSPOSE;
const mcsparseDirection_t dir = MCSPARSE_DIRECTION_COLUMN;

// 步骤 1：创建一个描述符，其中包含以下内容：
// - 矩阵 L 是从 1 开始编号的
// - 矩阵 L 是下三角矩阵
// - 矩阵 L 具有由参数 MCSPARSE_DIAG_TYPE_UNIT 指定的单位对角线
//   (L 可能并不具有所有对角线元素。)
mcsparseCreateMatDescr(&descr);
mcsparseSetMatIndexBase(descr, MCSPARSE_INDEX_BASE_ONE);
```

(下页继续)

(续上页)

```
mcsparseSetMatFillMode(descr, MCSPARSE_FILL_MODE_LOWER);
mcsparseSetMatDiagType(descr, MCSPARSE_DIAG_TYPE_UNIT);

// 步骤 2：创建一个空的信息结构体
mcsparseCreateBsrsv2Info(&info);

// 步骤 3：查询 bsrsv2 中使用了多少内存，并分配缓冲区
mcsparseDbsrsv2_bufferSize(handle, dir, trans, mb, nnzb, descr,
      d_bsrVal, d_bsrRowPtr, d_bsrColInd, blockDim, &pBufferSize);

// 由 mcMalloc 返回的 pBuffer 会自动对齐到 128 字节。
mcMalloc((void**)&pBuffer, pBufferSize);

// 步骤 4：执行分析
mcsparseDbsrsv2_analysis(handle, dir, trans, mb, nnzb, descr,
      d_bsrVal, d_bsrRowPtr, d_bsrColInd, blockDim,
      info, policy, pBuffer);
// L 具有单位对角线，因此不会报告结构性零。
status = mcsparseXbsrsv2_zeroPivot(handle, info, &structural_zero);
if (MCSPARSE_STATUS_ZERO_PIVOT == status){
   printf("L(%d,%d) is missing\n", structural_zero, structural_
↪zero);
}

// 步骤 5：处理 L*y = x
mcsparseDbsrsv2_solve(handle, dir, trans, mb, nnzb, &alpha, descr,
   d_bsrVal, d_bsrRowPtr, d_bsrColInd, blockDim, info,
   d_x, d_y, policy, pBuffer);
// L 具有单位对角线，因此不会报告任何数值零。
status = mcsparseXbsrsv2_zeroPivot(handle, info, &numerical_zero);
if (MCSPARSE_STATUS_ZERO_PIVOT == status){
   printf("L(%d,%d) is zero\n", numerical_zero, numerical_zero);
}

// 步骤 6：释放资源
mcFree(pBuffer);
mcsparseDestroyBsrsv2Info(info);
mcsparseDestroyMatDescr(descr);
mcsparseDestroy(handle);
```

输入

handle	处理 mcSPARSE 库上下文的句柄。
dir	块的存储格式，可以是 MCSPARSE_DIRECTION_ROW 或 MCSPARSE_DIRECTION_COLUMN。
transA	op(A) 操作
mb	矩阵 A 中块的行数和列数。
alpha	用于乘法运算的 <type> 标量。
descrA	矩阵 A 的描述符。支持的矩阵类型有 MCSPARSE_MATRIX_TYPE_GENERAL 此外，支持的索引基准有 MCSPARSE_INDEX_BASE_ZERO 和 MCSPARSE_INDEX_BASE_ONE。
bsrValA	存储矩阵 A 中 nnzb 个非零块的 <type> 数组。
bsrRowPtrA	包含每个块行的起始位置和最后一个块行结束位置后移一位的整数数组，数组长度为 mb。
bsrColIndA	存储了矩阵 A 中非零块 nnzb 个列索引信息的整数数组。
blockDim	稀疏矩阵 A 的块维度，大于零。
info	在分析阶段收集的结构信息 (应该未经修改地传递到求解阶段)。
x	大小为 m 的 <type> 右侧向量。
policy	支持的策略有 MCSPARSE_SOLVE_POLICY_NO_LEVEL 和 MCSPARSE_SOLVE_POLICY_USE_LEVEL。
pBuffer	用户分配的缓冲区，其大小由 bsrsv2_bufferSize() 返回。

输出

y	解向量的类型 <type>，大小为 m。

有关返回状态的描述，请参见mcsparseStatus_t。

8.6 mcsparseXbsrsv2_zeroPivot()

```
mcsparseStatus_t
mcsparseXbsrsv2_zeroPivot(mcsparseHandle_t handle,
                          bsrsv2Info_t     info,
                          int*             position)
```

如果返回的错误代码是 MCSPARSE_STATUS_ZERO_PIVOT，position=j 表示 A(j,j) 是结构零或数值零（奇异块）。否则为 position=-1。

position 可以从 0 开始，也可以从 1 开始，与矩阵相同。

函数 mcsparseXbsrsv2_zeroPivot() 是一个阻塞调用。它调用 mcDeviceSynchronize() 来确保所有先前的核都已完成。

position 可以位于主机内存或设备内存中。用户可以使用 mcsparseSetPointerMode() 设置正确的模式。

- 例程不需要额外的存储空间
- 如果流式有序内存分配器可用，则该例程支持异步执行

输入

handle	处理 mcSPARSE 库上下文的句柄。
info	如果用户已调用 bsrsv2_analysis() 或 bsrsv2_solve()，那么 info 包含结构零的或数值零。

输出

position	如果没有结构或数值为零，则position为-1；否则如果缺少A(j,j)或U(j,j)为零，则position=j。

有关返回状态的描述，请参见mcsparseStatus_t。

8.7 mcsparseCsrmvEx()

```
mcsparseStatus_t
mcsparseCsrmvEx_bufferSize(mcsparseHandle_t         handle,
                           mcsparseAlgMode_t        alg,
                           mcsparseOperation_t      transA,
                           int                      m,
                           int                      n,
                           int                      nnz,
                           const void*              alpha,
                           macaDataType             alphatype,
                           const mcsparseMatDescr_t descrA,
                           const void*              csrValA,
                           macaDataType             csrValAtype,
                           const int*               csrRowPtrA,
                           const int*               csrColIndA,
                           const void*              x,
                           macaDataType             xtype,
                           const void*              beta,
                           macaDataType             betatype,
                           void*                    y,
                           macaDataType             ytype,
                           macaDataType             executiontype,
                           size_t*                  bufferSizeInBytes)

mcsparseStatus_t
mcsparseCsrmvEx(mcsparseHandle_t         handle,
                mcsparseAlgMode_t        alg,
                mcsparseOperation_t      transA,
                int                      m,
                int                      n,
                int                      nnz,
                const void*              alpha,
                macaDataType             alphatype,
                const mcsparseMatDescr_t descrA,
                const void*              csrValA,
                macaDataType             csrValAtype,
                const int*               csrRowPtrA,
                const int*               csrColIndA,
                const void*              x,
                macaDataType             xtype,
                const void*              beta,
                macaDataType             betatype,
                void*                    y,
                macaDataType             ytype,
                macaDataType             executiontype,
                void*                    buffer)
```

此函数执行矩阵到向量的操作。

函数 mcsparseCsrmvEx_bufferSize 返回 mcsparseCsrmvEx 所需的工作区大小。

此函数具有以下限制：

- 所有指针都应该与 128 字节对齐
- 仅支持 MCSPARSE_OPERATION_NON_TRANSPOSE 操作
- 仅支持 MCSPARSE_MATRIX_TYPE_GENERAL 矩阵类型
- 仅支持 MCSPARSE_INDEX_BASE_ZERO 索引
- 不支持半精度
- 支持的最低 GPU 架构为 SM_53

此函数具有以下特性：

- 例程不需要额外的存储空间
- 例程支持异步执行

输入

alg	对于 csrmv 算法的实现，请参见mcsparseAlgMode_t 可能的取值。
alphatype	数据类型为 alpha。
csrValAtype	数据类型为 csrValA。
xtype	数据类型为 x。
betatype	数据类型为 beta。
ytype	数据类型为 y。
executiontype	用于计算的数据类型。
bufferSizeInBytes	指向 size_t 变量的指针，将被分配 mcsparseCsrmvEx 所需的工作区大小。
buffer	指向工作区缓冲区的指针。

有关返回状态的描述，请参见mcsparseStatus_t。

8.8 mcsparse<t>csrsv2_bufferSize()

```
mcsparseStatus_t
mcsparseScsrsv2_bufferSize(mcsparseHandle_t         handle,
                           mcsparseOperation_t      transA,
                           int                      m,
                           int                      nnz,
                           const mcsparseMatDescr_t descrA,
                           float*                   csrValA,
                           const int*               csrRowPtrA,
                           const int*               csrColIndA,
                           csrsv2Info_t             info,
                           int*                     pBufferSizeInBytes)

mcsparseStatus_t
mcsparseDcsrsv2_bufferSize(mcsparseHandle_t         handle,
                           mcsparseOperation_t      transA,
                           int                      m,
                           int                      nnz,
                           const mcsparseMatDescr_t descrA,
                           double*                  csrValA,
                           const int*               csrRowPtrA,
                           const int*               csrColIndA,
```

(下页继续)

(续上页)

```
                            csrsv2Info_t              info,
                            int*                      pBufferSizeInBytes)

mcsparseStatus_t
mcsparseCcsrsv2_bufferSize(mcsparseHandle_t          handle,
                            mcsparseOperation_t       transA,
                            int                       m,
                            int                       nnz,
                            const mcsparseMatDescr_t  descrA,
                            mcComplex*                csrValA,
                            const int*                csrRowPtrA,
                            const int*                csrColIndA,
                            csrsv2Info_t              info,
                            int*                      pBufferSizeInBytes)

mcsparseStatus_t
mcsparseZcsrsv2_bufferSize(mcsparseHandle_t          handle,
                            mcsparseOperation_t       transA,
                            int                       m,
                            int                       nnz,
                            const mcsparseMatDescr_t  descrA,
                            mcDoubleComplex*          csrValA,
                            const int*                csrRowPtrA,
                            const int*                csrColIndA,
                            csrsv2Info_t              info,
                            int*                      pBufferSizeInBytes)
```

此函数返回 csrsv2 (一种新的稀疏三角线性系统) 中使用的缓冲区大小。

虽然参数 trans 和 A 的上 (下) 三角部分有六种组合，但 csrsv2_bufferSize() 返回这些组合中所需的最大缓冲区大小。缓冲区大小取决于矩阵的维度和非零元素的数量。如果用户更改了矩阵，就需要再次调用 csrsv2_bufferSize() 来获得正确的缓冲区大小；否则，可能会发生分段错误。

- 例程不需要额外的存储空间
- 例程支持异步执行

输入

handle	处理 mcSPARSE 库上下文的句柄。
transA	操作 op(A)。
m	矩阵 A 的行数。
nnz	矩阵 A 的非零元素数。
descrA	矩阵 A 的描述符。支持的矩阵类型为 MCSPARSE_MATRIX_TYPE_GENERAL，而支持的对角线类型为 MCSPARSE_DIAG_TYPE_UNIT 和 MCSPARSE_DIAG_TYPE_NON_UNIT。
csrValA	类型为 <type> 的数组，其中包含矩阵 A 的 nnz 个非零元素。
csrRowPtrA	由整数组成的长度为 m + 1 的数组，其中包含每一行的起始位置和最后一行结束位置加一。
csrColIndA	整数数组，长度为 nnz，包含矩阵 A 非零元素的列索引。

输出

info	基于不同算法的内部状态记录。
pBufferSizeInBytes	在 csrsv2_analysis 和 csrsv2_solve 中使用的缓冲区的字节数。

有关返回状态的描述，请参见mcsparseStatus_t。

8.9 mcsparse<t>csrsv2_analysis()

```
mcsparseStatus_t
mcsparseScsrsv2_analysis(mcsparseHandle_t         handle,
                         mcsparseOperation_t      transA,
                         int                      m,
                         int                      nnz,
                         const mcsparseMatDescr_t descrA,
                         const float*             csrValA,
                         const int*               csrRowPtrA,
                         const int*               csrColIndA,
                         csrsv2Info_t             info,
                         mcsparseSolvePolicy_t    policy,
                         void*                    pBuffer)

mcsparseStatus_t
mcsparseDcsrsv2_analysis(mcsparseHandle_t         handle,
                         mcsparseOperation_t      transA,
                         int                      m,
                         int                      nnz,
                         const mcsparseMatDescr_t descrA,
                         const double*            csrValA,
                         const int*               csrRowPtrA,
                         const int*               csrColIndA,
                         csrsv2Info_t             info,
                         mcsparseSolvePolicy_t    policy,
                         void*                    pBuffer)

mcsparseStatus_t
mcsparseCcsrsv2_analysis(mcsparseHandle_t         handle,
                         mcsparseOperation_t      transA,
                         int                      m,
                         int                      nnz,
                         const mcsparseMatDescr_t descrA,
                         const mcComplex*         csrValA,
                         const int*               csrRowPtrA,
                         const int*               csrColIndA,
                         csrsv2Info_t             info,
                         mcsparseSolvePolicy_t    policy,
                         void*                    pBuffer)

mcsparseStatus_t
mcsparseZcsrsv2_analysis(mcsparseHandle_t         handle,
                         mcsparseOperation_t      transA,
                         int                      m,
                         int                      nnz,
                         const mcsparseMatDescr_t descrA,
                         const mcDoubleComplex*   csrValA,
                         const int*               csrRowPtrA,
                         const int*               csrColIndA,
                         csrsv2Info_t             info,
                         mcsparseSolvePolicy_t    policy,
                         void*                    pBuffer)
```

此函数执行新稀疏三角线性系统 csrsv2 的分析阶段。

对于给定的矩阵和特定的操作类型，预计此函数将仅执行一次。

此函数需要由 csrsv2_bufferSize() 返回的缓冲区大小。pBuffer 的地址必须是 128 字节的倍数。如果不是，则返回 MCSPARSE_STATUS_INVALID_VALUE。

函数 csrsv2_analysis() 报告一个结构零并计算存储在不透明结构 info 中的级别信息。层次信息可以为三角求解器提高更多的并行效率。但是，csrsv2_solve() 可以在没有级别信息的情况下完成。要禁用级别信息，用户需要将三角求解器的策略指定为 MCSPARSE_SOLVE_POLICY_NO_LEVEL。

函数 csrsv2_analysis() 始终报告第一个结构零，即使策略是 MCSPARSE_SOLVE_POLICY_NO_LEVEL。如果指定了 MCSPARSE_DIAG_TYPE_UNIT，即使某些 j 缺少 A(j,j)，也不会报告结构零。用户需要调用 mcsparseXcsrsv2_zeroPivot() 才能知道结构零的位置。

如果 csrsv2_analysis() 报告了一个结构零，则用户可以选择是否调用 csrsv2_solve()。在这种情况下，用户仍然可以调用 csrsv2_solve()，它将在与结构零相同的位置返回一个数值零。然而，结果 x 是没有意义的。

- 此功能需要内部分配的临时额外存储
- 如果流式有序内存分配器可用，则该例程支持异步执行

输入

handle	处理 mcSPARSE 库上下文的句柄。
transA	操作 op(A)。
m	矩阵 A 的行数。
nnz	矩阵 A 的非零元素数。
descrA	矩阵 A 的描述符。支持的矩阵类型为 MCSPARSE_MATRIX_TYPE_GENERAL，而支持的对角线类型为 MCSPARSE_DIAG_TYPE_UNIT 和 MCSPARSE_DIAG_TYPE_NON_UNIT。
csrValA	类型为 <type> 的数组，其中包含矩阵 A 的 nnz 个非零元素。
csrRowPtrA	由整数组成的长度为 m + 1 的数组，其中包含每一行的起始位置和最后一行结束位置加一。
csrColIndA	整数数组，长度为 nnz，包含矩阵 A 非零元素的列索引。
info	用 mcsparseCreateCsrsv2Info() 初始化的结构。
policy	支持的策略包括 MCSPARSE_SOLVE_POLICY_NO_LEVEL 和 HCSPARSE_SOLVE_POLICY_ USE_LEVEL。
pBuffer	缓冲区由用户分配，大小由 csrsv2_bufferSize() 返回。

输出

info	在分析阶段收集的信息填充的结构 (应在分析阶段传递给求解阶段而不变)。

有关返回状态的描述，请参见mcsparseStatus_t。

8.10 mcsparse<t>csrsv2_solve()

```
mcsparseStatus_t
mcsparseScsrsv2_solve(mcsparseHandle_t         handle,
                      mcsparseOperation_t      transA,
                      int                      m,
                      int                      nnz,
                      const float*             alpha,
                      const mcsparseMatDescr_t descra,
                      const float*             csrValA,
                      const int*               csrRowPtrA,
                      const int*               csrColIndA,
```

(下页继续)

(续上页)

```
                        csrsv2Info_t              info,
                        const float*              x,
                        float*                    y,
                        mcsparseSolvePolicy_t     policy,
                        void*                     pBuffer)

mcsparseStatus_t
mcsparseDcsrsv2_solve(mcsparseHandle_t          handle,
                        mcsparseOperation_t       transA,
                        int                       m,
                        int                       nnz,
                        const double*             alpha,
                        const mcsparseMatDescr_t  descra,
                        const double*             csrValA,
                        const int*                csrRowPtrA,
                        const int*                csrColIndA,
                        csrsv2Info_t              info,
                        const double*             x,
                        double*                   y,
                        mcsparseSolvePolicy_t     policy,
                        void*                     pBuffer)

mcsparseStatus_t
mcsparseCcsrsv2_solve(mcsparseHandle_t          handle,
                        mcsparseOperation_t       transA,
                        int                       m,
                        int                       nnz,
                        const mcComplex*          alpha,
                        const mcsparseMatDescr_t  descra,
                        const mcComplex*          csrValA,
                        const int*                csrRowPtrA,
                        const int*                csrColIndA,
                        csrsv2Info_t              info,
                        const mcComplex*          x,
                        mcComplex*                y,
                        mcsparseSolvePolicy_t     policy,
                        void*                     pBuffer)

mcsparseStatus_t
mcsparseZcsrsv2_solve(mcsparseHandle_t          handle,
                        mcsparseOperation_t       transA,
                        int                       m,
                        int                       nnz,
                        const mcDoubleComplex*    alpha,
                        const mcsparseMatDescr_t  descra,
                        const mcDoubleComplex*    csrValA,
                        const int*                csrRowPtrA,
                        const int*                csrColIndA,
                        csrsv2Info_t              info,
                        const mcDoubleComplex*    x,
                        mcDoubleComplex*          y,
                        mcsparseSolvePolicy_t     policy,
                        void*                     pBuffer)
```

此函数执行 csrsv2 的求解阶段，这是一个新的稀疏三角线性系统。

对于给定矩阵和特定操作类型，可以多次执行此函数。

此函数需要由 csrsv2_bufferSize() 返回缓冲区大小。pBuffer 的地址必须是 128 字节的倍数。如果不是，则返回 MCSPARSE_STATUS_INVALID_VALUE。

尽管可以在没有级别信息的情况下完成 csrsv2_solve()，但用户仍然需要注意一致性。如果使用策略 MCSPARSE_SOLVE_POLICY_USE_LEVEL 调用 csrsv2_analysis()，则可以在有或没有级别的情况下运行 csrsv2_solve()。相反，如果使用 MCSPARSE_SOLVE_POLICY_NO_LEVEL 调用 csrsv2_analysis()，则 csrsv2_solve() 只能接受 MCSPARSE_SOLVE_POLICY_NO_LEVEL。否则，返回 MCSPARSE_STATUS_INVALID_VALUE。

级别信息可能不会提高性能，但会花费额外的时间进行分析。例如，一个三对角矩阵没有并行性。在这种情况下，MCSPARSE_SOLVE_POLICY_NO_LEVEL 比 MCSPARSE_SOLVE_POLICY_USE_LEVEL 性能更好。如果用户有一个迭代求解器，最好的方法是用 MCSPARSE_SOLVE_POLICY_USE_LEVEL 执行一次 csrsv2_analysis()。然后在第一次运行中使用 MCSPARSE_SOLVE_POLICY_NO_LEVEL 执行 csrsv2_solve() 函数，在第二次运行中使用 MCSPARSE_SOLVE_POLICY_USE_LEVEL，选择更快的一个函数来执行剩余的迭代。

函数 csrsv2_solve() 会报告第一个数值零，包括结构零。如果 status 为 0，则表示没有找到任何数值零。此外，如果指定了 MCSPARSE_DIAG_TYPE_UNIT，则即使某些 j 对应的 A(j,j) 为零，也不会报告任何数值零。用户需要调用 mcsparseXcsrsv2_zeroPivot() 才能知道数值零的位置。

例如，假设 L 是一个具有单位对角线的下三角矩阵，下面的代码通过级别信息求解 L*y=x。

```
// 假设 L 是一个用 CSR 格式表示的 m x m 的稀疏矩阵。
// L 是具有单位对角线的下三角矩阵。
// 假设：
// - 矩阵 L 的维度为 m，
// - 矩阵 L 拥有 nnz 个零元素，
// - 句柄已经由 mcsparseCreate() 创建，
// - (d_csrRowPtr，d_csrColInd，d_csrVal) 用设备内存上矩阵 L 的 CSR 格
式表示，
// - d_x 是设备内存上的右侧向量，
// - d_y 是设备内存上的解向量。

mcsparseMatDescr_t descr = 0;
csrsv2Info_t info = 0;
int pBufferSize;
void *pBuffer = 0;
int structural_zero;
int numerical_zero;
const double alpha = 1.;
const mcsparseSolvePolicy_t policy = MCSPARSE_SOLVE_POLICY_USE_
↪LEVEL;
const mcsparseOperation_t trans = MCSPARSE_OPERATION_NON_TRANSPOSE;

// 步骤 1：创建包含以下内容的描述符
// - 矩阵 L 以 1 为基数
// - 矩阵 L 为下三角形
// - 矩阵 L 具有单位对角线，由参数 MCSPARSE_DIAG_TYPE_UNIT 指定
//   (L 可能没有所有对角线元素。)
mcsparseCreateMatDescr(&descr);
mcsparseSetMatIndexBase(descr, MCSPARSE_INDEX_BASE_ONE);
mcsparseSetMatFillMode(descr, MCSPARSE_FILL_MODE_LOWER);
mcsparseSetMatDiagType(descr, MCSPARSE_DIAG_TYPE_UNIT);

// 步骤 2：创建一个空的 info 结构
mcsparseCreateCsrsv2Info(&info);

// 步骤 3：查询 csrsv2 中使用了多少内存，并分配缓冲区
mcsparseDcsrsv2_bufferSize(handle, trans, m, nnz, descr,
```

(下页继续)

(续上页)

```
      d_csrVal, d_csrRowPtr, d_csrColInd, info, &pBufferSize);
// 由 mcMalloc 返回的 pBuffer 会自动对齐到 128 字节。
mcMalloc((void**)&pBuffer, pBufferSize);

// 步骤 4：执行分析
mcsparseDcsrsv2_analysis(handle, trans, m, nnz, descr,
      d_csrVal, d_csrRowPtr, d_csrColInd,
      info, policy, pBuffer);
// L 具有单位对角线，因此不会报告任何结构零的位置。
status = mcsparseXcsrsv2_zeroPivot(handle, info, &structural_zero);
if (MCSPARSE_STATUS_ZERO_PIVOT == status){
   printf("L(%d,%d) is missing\n", structural_zero, structural_
↪zero);
}

// 步骤 5：求解 L*y = x
mcsparseDcsrsv2_solve(handle, trans, m, nnz, &alpha, descr,
   d_csrVal, d_csrRowPtr, d_csrColInd, info,
   d_x, d_y, policy, pBuffer);
// L 具有单位对角线，因此不会报告任何数值零的位置。
status = mcsparseXcsrsv2_zeroPivot(handle, info, &numerical_zero);
if (MCSPARSE_STATUS_ZERO_PIVOT == status){
   printf("L(%d,%d) is zero\n", numerical_zero, numerical_zero);
}

// 步骤 6：释放资源
mcFree(pBuffer);
mcsparseDestroyCsrsv2Info(info);
mcsparseDestroyMatDescr(descr);
mcsparseDestroy(handle);
```

注解： `csrsv2_solve()` 每行需要更多的非零元素才能达到良好的性能。如果平均每行超过 16 个非零元素，它的性能会更好。

如果 `pBuffer != NULL`，则此函数将支持以下属性

- 例程不需要额外的存储空间
- 例程支持异步执行

输入

handle	处理 mcSPARSE 库上下文的句柄。
transA	操作 op(A)。
m	矩阵 A 的行数和列数。
alpha	用于乘法的 <type> 标量。
descrA	矩阵 A 的描述符。支持的矩阵类型为 MCSPARSE_MATRIX_TYPE_GENERAL，而支持的对角线类型为 MCSPARSE_DIAG_TYPE_UNIT 和 MCSPARSE_DIAG_TYPE_NON_UNIT。
csrValA	类型为 <type> 的数组，其中包含矩阵 A 的 nnz 个非零元素。
csrRowPtrA	由整数组成的长度为 m + 1 的数组，其中包含每一行的起始位置和最后一行结束位置加一。
csrColIndA	整数数组，包含矩阵 A 中 nnz 个非零元素的列索引。
info	在分析阶段收集的信息的结构 (应该不经更改地传递到求解阶段)。
x	大小为 m 的 <type> 右侧向量。
policy	支持的策略包括 MCSPARSE_SOLVE_POLICY_NO_LEVEL 和 MCSPARSE_SOLVE_POLICY_ USE_LEVEL。
pBuffer	缓冲区由用户分配，大小由 csrsv2_bufferSize() 返回。

输出

y	大小为 m 的 <type> 解向量

有关返回状态的描述，请参见mcsparseStatus_t。

8.11 mcsparseXcsrsv2_zeroPivot()

```
mcsparseStatus_t
mcsparseXcsrsv2_zeroPivot(mcsparseHandle_t handle,
                          csrsv2Info_t     info,
                          int*             position)
```

如果返回的错误代码为 MCSPARSE_STATUS_ZERO_PIVOT，position=j 表示 A(j,j) 存在结构零或数值零。否则 position=-1。

position 可以从 0 开始，也可以从 1 开始，与矩阵相同。

函数 mcsparseXcsrsv2_zeroPivot() 是一个阻塞调用。它调用 mcDeviceSynchronize() 来确保所有先前的核都已完成。

position 可以位于主机内存或设备内存中。用户可以使用 mcsparseSetPointerMode() 设置正确的模式。

- 例程不需要额外的存储空间
- 如果流式有序内存分配器可用，则该例程支持异步执行

输入

handle	处理 mcSPARSE 库上下文的句柄。
info	如果用户已调用 csrsv2_analysis() 或 csrsv2_solve() info 包含结构零或数值零

输出

position	如果没有结构零或数值零，则 position 为 -1；否则如果缺少 A(j,j) 或 U(j,j) 为零，则 position=j。

有关返回状态的描述，请参见mcsparseStatus_t。

8.12 mcsparse<t>gemvi()

```
mcsparseStatus_t
mcsparseSgemvi_bufferSize(mcsparseHandle_t    handle,
                          mcsparseOperation_t transA,
                          int                 m,
                          int                 n,
                          int                 nnz,
                          int*                pBufferSize)

mcsparseStatus_t
mcsparseDgemvi_bufferSize(mcsparseHandle_t    handle,
                          mcsparseOperation_t transA,
                          int                 m,
                          int                 n,
                          int                 nnz,
                          int*                pBufferSize)

mcsparseStatus_t
mcsparseCgemvi_bufferSize(mcsparseHandle_t    handle,
                          mcsparseOperation_t transA,
                          int                 m,
                          int                 n,
                          int                 nnz,
                          int*                pBufferSize)

mcsparseStatus_t
mcsparseZgemvi_bufferSize(mcsparseHandle_t    handle,
                          mcsparseOperation_t transA,
                          int                 m,
                          int                 n,
                          int                 nnz,
                          int*                pBufferSize)
```

```
mcsparseStatus_t
mcsparseSgemvi(mcsparseHandle_t     handle,
               mcsparseOperation_t  transA,
               int                  m,
               int                  n,
               const float*         alpha,
               const float*         A,
               int                  lda,
               int                  nnz,
               const float*         x,
               const int*           xInd,
               const float*         beta,
               float*               y,
               mcsparseIndexBase_t  idxBase,
               void*                pBuffer)

mcsparseStatus_t
mcsparseDgemvi(mcsparseHandle_t     handle,
               mcsparseOperation_t  transA,
```

(下页继续)

(续上页)

```
                int                   m,
                int                   n,
                const double*         alpha,
                const double*         A,
                int                   lda,
                int                   nnz,
                const double*         x,
                const int*            xInd,
                const float*          beta,
                double*               y,
                mcsparseIndexBase_t   idxBase,
                void*                 pBuffer)

mcsparseStatus_t
mcsparseCgemvi(mcsparseHandle_t     handle,
                mcsparseOperation_t  transA,
                int                  m,
                int                  n,
                const mcComplex*     alpha,
                const mcComplex*     A,
                int                  lda,
                int                  nnz,
                const mcComplex*     x,
                const int*           xInd,
                const float*         beta,
                mcComplex*           y,
                mcsparseIndexBase_t  idxBase,
                void*                pBuffer)

mcsparseStatus_t
mcsparseZgemvi(mcsparseHandle_t       handle,
                mcsparseOperation_t    transA,
                int                    m,
                int                    n,
                const mcDoubleComplex* alpha,
                const mcDoubleComplex* A,
                int                    lda,
                int                    nnz,
                const mcDoubleComplex* x,
                const int*             xInd,
                const float*           beta,
                mcDoubleComplex*       y,
                mcsparseIndexBase_t    idxBase,
                void*                  pBuffer)
```

此函数执行矩阵到向量的操作。

- 例程不需要额外的存储空间
- 例程支持异步执行

函数 `mcsparse<t>gemvi_bufferSize()` 返回 `mcsparse<t>gemvi()` 中使用的缓冲区大小。

输入

handle	处理 mcSPARSE 库上下文的句柄。
trans	操作 op(A)
m	矩阵 A 的行数。
n	矩阵 A 的列数。
alpha	用于乘法的 <type> 标量。
A	指向稠密矩阵 A 的指针。
lda	A 的主维度的大小。
nnz	向量 x 的非零元素数。
x	大小为 n 的 <type> 稀疏向量，具有 nnz 个非零元素。
xInd	x 中非零值的索引
beta	用于乘法的 <type> 标量。如果 beta 为零，y 不需要是一个有效的输入。
y	包含 m 个元素的稠密向量。
idxBase	0 或 1，分别表示基于 0 或基于 1 的索引。
pBufferSize	在 mcsparse<t>gemvi() 函数中使用的缓冲区所需的元素数量。
pBuffer	工作空间缓冲区

输出

y	更新后的 <type> 稠密向量

有关返回状态的描述，请参见mcsparseStatus_t。

9 mcSPARSE 3 级函数参考

本章介绍稀疏线性代数函数，这些函数在稀疏矩阵和（通常很高的）稠密矩阵之间执行运算。

具有多个右侧值的稀疏三角线性系统的求解分两阶段实现。首先，在分析阶段，通过调用适当的 `csrsm2_analysis()` 函数，分析稀疏三角矩阵以确定其元素之间的依赖关系。该分析特定于给定矩阵的稀疏性模式和选定的 `mcsparseOperation_t` 类型。来自分析阶段的信息存储在类型 `csrsm2Info_t` 的参数中，该参数之前已通过调用 `mcsparseCreateCsrsm2Info()` 进行初始化。

其次，在求解阶段，通过调用适当的 `csrsm2_solve()` 函数，使用存储在 `csrsm2Info_t` 参数中的信息求解给定的稀疏三角线性系统。求解阶段可以使用不同的多个右侧值多次执行，而分析阶段只需要执行一次。当一次需要逐个求解不同集合的多个右侧值时，稀疏三角线性系统的解法尤其有用，同时其系数矩阵保持不变。

最后，一旦所有求解完成，可以通过调用 `mcsparseDestroyCsrsm2Info()` 来释放由 `csrsm2Info_t` 参数指向的不透明数据结构。

9.1 mcsparse<t>bsrmm()

```
mcsparseStatus_t
mcsparseSbsrmm(mcsparseHandle_t          handle,
               mcsparseDirection_t       dirA,
               mcsparseOperation_t       transA,
               mcsparseOperation_t       transB,
               int                       mb,
               int                       n,
               int                       kb,
               int                       nnzb,
               const float*              alpha,
               const mcsparseMatDescr_t  descrA,
               const float*              bsrValA,
               const int*                bsrRowPtrA,
               const int*                bsrColIndA,
               int                       blockDim,
               const float*              B,
               int                       ldb,
               const float*              beta,
               float*                    C,
               int                       ldc)

mcsparseStatus_t
mcsparseDbsrmm(mcsparseHandle_t          handle,
               mcsparseDirection_t       dirA,
               mcsparseOperation_t       transA,
               mcsparseOperation_t       transB,
               int                       mb,
```

(下页继续)

(续上页)

```
                int                         n,
                int                         kb,
                int                         nnzb,
                const double*               alpha,
                const mcsparseMatDescr_t    descrA,
                const double*               bsrValA,
                const int*                  bsrRowPtrA,
                const int*                  bsrColIndA,
                int                         blockDim,
                const double*               B,
                int                         ldb,
                const double*               beta,
                double*                     C,
                int                         ldc)

mcsparseStatus_t
mcsparseCbsrmm(mcsparseHandle_t             handle,
                mcsparseDirection_t         dirA,
                mcsparseOperation_t         transA,
                mcsparseOperation_t         transB,
                int                         mb,
                int                         n,
                int                         kb,
                int                         nnzb,
                const mcComplex*            alpha,
                const mcsparseMatDescr_t    descrA,
                const mcComplex*            bsrValA,
                const int*                  bsrRowPtrA,
                const int*                  bsrColIndA,
                int                         blockDim,
                const mcComplex*            B,
                int                         ldb,
                const mcComplex*            beta,
                mcComplex*                  C,
                int                         ldc)

mcsparseStatus_t
mcsparseZbsrmm(mcsparseHandle_t             handle,
                mcsparseDirection_t         dirA,
                mcsparseOperation_t         transA,
                mcsparseOperation_t         transB,
                int                         mb,
                int                         n,
                int                         kb,
                int                         nnzb,
                const mcDoubleComplex*      alpha,
                const mcsparseMatDescr_t    descrA,
                const mcDoubleComplex*      bsrValA,
                const int*                  bsrRowPtrA,
                const int*                  bsrColIndA,
                int                         blockDim,
                const mcDoubleComplex*      B,
                int                         ldb,
                const mcDoubleComplex*      beta,
                mcDoubleComplex*            C,
                int                         ldc)
```

此函数具有以下限制：

- 仅支持 MCSPARSE_MATRIX_TYPE_GENERAL 矩阵类型
- 仅支持 blockDim > 1

transpose(B) 的目的是改善矩阵 B 的内存访问。矩阵 B 按列主顺序排列的 A*transpose(B) 的计算模式等效于矩阵 B 按行主顺序排列的 A*B。

实际上，迭代求解器或特征值求解器中的任何操作都不会使用 A*transpose(B)。但是，我们可以执行与 A*B 相同的 A*transpose(transpose(B))。例如，假设 A 是 mb*kb，B 是 k*n，C 是 m*n，下面的代码显示了 mcsparseDbsrmm() 的用法。

```
// A 是 mb*kb，B 是 k*n，C 是 m*n
      const int m = mb*blockSize;
      const int k = kb*blockSize;
      const int ldb_B = k; // B 的前导维度
      const int ldc   = m; // C 的前导维度
// 执行 C: =alpha*A*B + beta*C
      mcsparseSetMatType(descrA, MCSPARSE_MATRIX_TYPE_GENERAL );
      mcsparseDbsrmm(mcsparse_handle,
 MCSPARSE_DIRECTION_COLUMN, MCSPARSE_OPERATION_NON_TRANSPOSE,
↪MCSPARSE_OPERATION_NON_TRANSPOSE, mb, n, kb, nnzb, alpha, descrA,
↪ bsrValA, bsrRowPtrA, bsrColIndA, blockSize, B, ldb_B, beta, C,
↪ldc);
```

我们的建议不是使用 A*B，而是首先通过调用 mcblas<t>geam() 将 B 转置为 Bt，然后执行 A*transpose(Bt)。

```
// 步骤 1: Bt: = transpose(B)
      const int m = mb*blockSize;
      const int k = kb*blockSize;
      double *Bt;
      const int ldb_Bt = n; // Bt 的前导维度
      mcMalloc((void**)&Bt, sizeof(double)*ldb_Bt*k);
      double one  = 1.0;
      double zero = 0.0;
      mcblasSetPointerMode(mcblas_handle, MCBLAS_POINTER_MODE_
↪HOST);
      mcblasDgeam(mcblas_handle, MCBLAS_OP_T, MCBLAS_OP_T,
         n, k, &one, B, int ldb_B, &zero, B, int ldb_B, Bt, ldb_
↪Bt);

// 步骤 2: 执行 C: =alpha*A*transpose(Bt) + beta*C
      mcsparseDbsrmm(mcsparse_handle,
      MCSPARSE_DIRECTION_COLUMN, MCSPARSE_OPERATION_NON_TRANSPOSE,
      MCSPARSE_OPERATION_TRANSPOSE, mb, n, kb, nnzb, alpha, descrA,
      bsrValA, bsrRowPtrA, bsrColIndA, blockSize, Bt, ldb_Bt, beta,
↪ C, ldc);
```

bsrmm() 具有以下特性：

- 例程不需要额外的存储空间
- 例程支持异步执行

输入

handle	处理 mcSPARSE 库上下文的句柄。
dir	块的存储格式，MCSPARSE_DIRECTION_ROW 或 MCSPARSE_DIRECTION_COLUMN。
transA	操作 op(A)。
transB	操作 op(B)。
mb	稀疏矩阵 A 的块行数。
n	稠密矩阵 op(B) 和 A 的列数。
kb	稀疏矩阵 A 的块列数。
nnzb	稀疏矩阵 A 的非零块的数量。
alpha	用于乘法的 <type> 标量。
descrA	矩阵 A 的描述符。支持的矩阵类型为 MCSPARSE_MATRIX_TYPE_GENERAL 此外，支持的索引库为 MCSPARSE_INDEX_BASE_ZERO 和 MCSPARSE_INDEX_BASE_ONE。
bsrValA	包含矩阵 A 的 nnzb 个非零块的 <type> 数组。
bsrRowPtrA	由整数组成的长度为 mb + 1 的数组，其中包含每个块行的起始位置和最后一个块行结束位置加一。
bsrColIndA	包含矩阵 A 的 nnzb 个非零块的列索引的整数数组。
blockDim	稀疏矩阵 A 的块维度，大于零。
B	如果 op(B)=B，则为 (ldb,n) 维度的数组；否则为 (ldb,k) 维度的数组。
ldb	B 的主维度。
beta	用于乘法的 <type> 标量。如果 beta 为零，则 C 不必是有效输入。
C	维度为 (ldc, n) 的数组。
ldc	C 的主维度。

输出

C	更新后的维度为 (ldc, n) 的 <type> 数组。

有关返回状态的描述，请参见mcsparseStatus_t。

9.2 mcsparse<t>csrsm2_bufferSizeExt()

```
mcsparseStatus_t
mcsparseScsrsm2_bufferSizeExt(mcsparseHandle_t          handle,
                              int                       algo,
                              mcsparseOperation_t       transA,
                              mcsparseOperation_t       transB,
                              int                       m,
                              int                       nrhs,
                              int                       nnz,
                              const float*              alpha,
                              const mcsparseMatDescr_t  descrA,
                              const float*              csrSortedValA,
                              const int*                csrSortedRowPtrA,
                              const int*                csrSortedColIndA,
                              const float*              B,
                              int                       ldb,
                              csrsm2Info_t              info,
                              mcsparseSolvePolicy_t     policy,
                              size_t*                   pBufferSize)

mcsparseStatus_t
mcsparseDcsrsm2_bufferSizeExt(mcsparseHandle_t          handle,
```

(下页继续)

(续上页)

```
                                int                         algo,
                                mcsparseOperation_t         transA,
                                mcsparseOperation_t         transB,
                                int                         m,
                                int                         nrhs,
                                int                         nnz,
                                const double*               alpha,
                                const mcsparseMatDescr_t    descrA,
                                const double*               csrSortedValA,
                                const int*                  csrSortedRowPtrA,
                                const int*                  csrSortedColIndA,
                                const double*               B,
                                int                         ldb,
                                csrsm2Info_t                info,
                                mcsparseSolvePolicy_t       policy,
                                size_t*                     pBufferSize)

mcsparseStatus_t
mcsparseCcsrsm2_bufferSizeExt(mcsparseHandle_t          handle,
                                int                         algo,
                                mcsparseOperation_t         transA,
                                mcsparseOperation_t         transB,
                                int                         m,
                                int                         nrhs,
                                int                         nnz,
                                const mcComplex*            alpha,
                                const mcsparseMatDescr_t    descrA,
                                const mcComplex*            csrSortedValA,
                                const int*                  csrSortedRowPtrA,
                                const int*                  csrSortedColIndA,
                                const mcComplex*            B,
                                int                         ldb,
                                csrsm2Info_t                info,
                                mcsparseSolvePolicy_t       policy,
                                size_t*                     pBufferSize)

mcsparseStatus_t
mcsparseZcsrsm2_bufferSizeExt(mcsparseHandle_t          handle,
                                int                         algo,
                                mcsparseOperation_t         transA,
                                mcsparseOperation_t         transB,
                                int                         m,
                                int                         nrhs,
                                int                         nnz,
                                const mcDoubleComplex*      alpha,
                                const mcsparseMatDescr_t    descrA,
                                const mcDoubleComplex*      csrSortedValA,
                                const int*                  csrSortedRowPtrA,
                                const int*                  csrSortedColIndA,
                                const mcDoubleComplex*      B,
                                int                         ldb,
                                csrsm2Info_t                info,
                                mcsparseSolvePolicy_t       policy,
                                size_t*                     pBufferSize)
```

此函数返回在稀疏三角线性系统 `csrsm2` 中使用的缓冲区的大小。

- 例程不需要额外的存储空间
- 例程支持异步执行

输入

handle	处理 mcSPARSE 库上下文的句柄。
algo	algo = 0 是非阻塞版本。algo = 1 是阻塞版本。
transA	操作 op(A)。
transB	操作 op(B)。
m	矩阵 A 的行数。
nrhs	右侧矩阵 op(B) 的列数。
nnz	矩阵 A 的非零元素数。
alpha	用于乘法的 <type> 标量。
descrA	矩阵 A 的描述符。支持的矩阵类型为 MCSPARSE_MATRIX_TYPE_GENERAL，而支持的对角线类型为 MCSPARSE_DIAG_TYPE_UNIT 和 MCSPARSE_DIAG_TYPE_NON_UNIT。
csrValA	类型为 <type> 的数组，其中包含矩阵 A 中 nnz 个非零元素。
csrRowPtrA	由整数组成的长度为 m + 1 的数组，其中包含每一行的起始位置和最后一行结束位置加一。
csrColIndA	整数数组，长度为 nnz，包含矩阵 A 非零元素的列索引。
B	<type> 的右侧矩阵。op(B) 的大小为 m-by-nrhs。
ldb	B 和 X 的前导维度。
info	在分析阶段收集的信息的结构 (应该不经更改地传递到求解阶段)。
policy	支持的策略包括 MCSPARSE_SOLVE_POLICY_NO_LEVEL 和 MCSPARSE_SOLVE_POLICY_USE_LEVEL。

输出

info	基于不同算法的内部状态记录。
pBufferSize	在 csrsm2_analysis 和 csrsm2_solve 使用的字节数。

有关返回状态的描述，请参见mcsparseStatus_t。

9.3 mcsparse<t>csrsm2_analysis()

```
mcsparseStatus_t
mcsparseScsrsm2_analysis(mcsparseHandle_t          handle,
                         int                       algo,
                         mcsparseOperation_t       transA,
                         mcsparseOperation_t       transB,
                         int                       m,
                         int                       nrhs,
                         int                       nnz,
                         const float*              alpha,
                         const mcsparseMatDescr_t  descrA,
                         const float*              csrSortedValA,
                         const int*                csrSortedRowPtrA,
                         const int*                csrSortedColIndA,
                         const float*              B,
                         int                       ldb,
                         csrsm2Info_t              info,
                         mcsparseSolvePolicy_t     policy,
                         void*                     pBuffer)
```

(下页继续)

(续上页)

```
mcsparseStatus_t
mcsparseDcsrsm2_analysis(mcsparseHandle_t         handle,
                         int                      algo,
                         mcsparseOperation_t      transA,
                         mcsparseOperation_t      transB,
                         int                      m,
                         int                      nrhs,
                         int                      nnz,
                         const double*            alpha,
                         const mcsparseMatDescr_t descrA,
                         const double*            csrSortedValA,
                         const int*               csrSortedRowPtrA,
                         const int*               csrSortedColIndA,
                         const double*            B,
                         int                      ldb,
                         csrsm2Info_t             info,
                         mcsparseSolvePolicy_t    policy,
                         void*                    pBuffer)

mcsparseStatus_t
mcsparseCcsrsm2_analysis(mcsparseHandle_t         handle,
                         int                      algo,
                         mcsparseOperation_t      transA,
                         mcsparseOperation_t      transB,
                         int                      m,
                         int                      nrhs,
                         int                      nnz,
                         const mcComplex*         alpha,
                         const mcsparseMatDescr_t descrA,
                         const mcComplex*         csrSortedValA,
                         const int*               csrSortedRowPtrA,
                         const int*               csrSortedColIndA,
                         const mcComplex*         B,
                         int                      ldb,
                         csrsm2Info_t             info,
                         mcsparseSolvePolicy_t    policy,
                         void*                    pBuffer)

mcsparseStatus_t
mcsparseZcsrsm2_analysis(mcsparseHandle_t         handle,
                         int                      algo,
                         mcsparseOperation_t      transA,
                         mcsparseOperation_t      transB,
                         int                      m,
                         int                      nrhs,
                         int                      nnz,
                         const mcDoubleComplex*   alpha,
                         const mcsparseMatDescr_t descrA,
                         const mcDoubleComplex*   csrSortedValA,
                         const int*               csrSortedRowPtrA,
                         const int*               csrSortedColIndA,
                         const mcDoubleComplex*   B,
                         int                      ldb,
                         csrsm2Info_t             info,
                         mcsparseSolvePolicy_t    policy,
```

(下页继续)

(续上页)

```
                                    void*                   pBuffer)
```

此函数执行稀疏三角线性系统 csrsm2 的分析阶段。

预期此函数在给定的矩阵和特定操作类型下只执行一次。

此函数需要由 csrsm2_bufferSize() 返回的缓冲区大小。pBuffer 的地址必须是 128 字节的倍数。如果不满足这些条件，将返回 MCSPARSE_STATUS_INVALID_VALUE。

函数 csrsm2_analysis() 报告一个结构零，并计算存储在不透明结构 info 中的级别信息。级别信息可以提高三角求解器的更多并行效率。然而，csrsm2_solve() 可以在没有级别信息的情况下执行。要禁用级别信息，用户需要将三角求解器的策略指定为 MCSPARSE_SOLVE_POLICY_NO_LEVEL。

无论策略是 MCSPARSE_SOLVE_POLICY_NO_LEVEL 还是其他，函数 csrsm2_analysis() 总是报告第一个结构零。如果指定了 MCSPARSE_DIAG_TYPE_UNIT，并且对于某些 j，A(j,j) 缺失，则不会报告结构零。用户需要调用 mcsparseXcsrsm2_zeroPivot() 来获取结构零的位置。

如果 csrsm2_analysis() 报告一个结构零，用户可自行决定是否调用 csrsm2_solve()。

在这种情况下，用户仍然可以调用 csrsm2_solve()，此函数将在与结构零相同的位置返回一个数值零。结果 X 是没有意义的。

- 此函数需要内部分配的临时额外存储空间。
- 如果流式有序内存分配器可用，则该例程支持异步执行

输入

handle	处理 mcSPARSE 库上下文的句柄。
algo	algo = 0 是非阻塞版本。algo = 1 是阻塞版本。
transA	操作 op(A)。
transB	操作 op(B)。
m	矩阵 A 的行数。
nrhs	右侧矩阵 op(B) 的列数。
nnz	矩阵 A 的非零元素数。
alpha	用于乘法的 <type> 标量。
descrA	矩阵 A 的描述符。支持的矩阵类型为 MCSPARSE_MATRIX_TYPE_GENERAL，而支持的对角线类型为 MCSPARSE_DIAG_TYPE_UNIT 和 MCSPARSE_DIAG_TYPE_NON_UNIT。
csrValA	类型为 <type> 的数组，其中包含矩阵 A 中 nnz 个非零元素。
csrRowPtrA	由整数组成的长度为 m + 1 的数组，其中包含每一行的起始位置和最后一行结束位置加一。
csrColIndA	整数数组，长度为 nnz，包含矩阵 A 非零元素的列索引。
B	<type> 的右侧矩阵。op(B) 的大小为 m-by-nrhs
ldb	B 和 X 的前导维度。
info	在分析阶段收集的信息的结构 (应该不经更改地传递到求解阶段)。
policy	支持的策略包括 MCSPARSE_SOLVE_POLICY_NO_LEVEL 和 MCSPARSE_SOLVE_POLICY_USE_LEVEL
pBuffer	用户分配的缓冲区，大小由 csrsm2_bufferSize() 返回。

输出

info	info 是一个在分析阶段收集信息的结构体，(在求解阶段不改变并传递给函数使用)。

有关返回状态的描述，请参见mcsparseStatus_t。

9.4 mcsparse<t>csrsm2_solve()

```
mcsparseStatus_t
mcsparseScsrsm2_solve(mcsparseHandle_t         handle,
                      int                      algo,
                      mcsparseOperation_t      transA,
                      mcsparseOperation_t      transB,
                      int                      m,
                      int                      nrhs,
                      int                      nnz,
                      const float*             alpha,
                      const mcsparseMatDescr_t descrA,
                      const float*             csrSortedValA,
                      const int*               csrSortedRowPtrA,
                      const int*               csrSortedColIndA,
                      float*                   B,
                      int                      ldb,
                      csrsm2Info_t             info,
                      mcsparseSolvePolicy_t    policy,
                      void*                    pBuffer)

mcsparseStatus_t
mcsparseDcsrsm2_solve(mcsparseHandle_t         handle,
                      int                      algo,
                      mcsparseOperation_t      transA,
                      mcsparseOperation_t      transB,
                      int                      m,
                      int                      nrhs,
                      int                      nnz,
                      const double*            alpha,
                      const mcsparseMatDescr_t descrA,
                      const double*            csrSortedValA,
                      const int*               csrSortedRowPtrA,
                      const int*               csrSortedColIndA,
                      double*                  B,
                      int                      ldb,
                      csrsm2Info_t             info,
                      mcsparseSolvePolicy_t    policy,
                      void*                    pBuffer)

mcsparseStatus_t
mcsparseCcsrsm2_solve(mcsparseHandle_t         handle,
                      int                      algo,
                      mcsparseOperation_t      transA,
                      mcsparseOperation_t      transB,
                      int                      m,
                      int                      nrhs,
                      int                      nnz,
                      const mcComplex*         alpha,
                      const mcsparseMatDescr_t descrA,
                      const mcComplex*         csrSortedValA,
                      const int*               csrSortedRowPtrA,
                      const int*               csrSortedColIndA,
                      mcComplex*               B,
                      int                      ldb,
                      csrsm2Info_t             info,
```

(下页继续)

(续上页)

```
                        mcsparseSolvePolicy_t    policy,
                        void*                    pBuffer)

mcsparseStatus_t
mcsparseZcsrsm2_solve(mcsparseHandle_t         handle,
                        int                      algo,
                        mcsparseOperation_t      transA,
                        mcsparseOperation_t      transB,
                        int                      m,
                        int                      nrhs,
                        int                      nnz,
                        const mcDoubleComplex*   alpha,
                        const mcsparseMatDescr_t descrA,
                        const mcDoubleComplex*   csrSortedValA,
                        const int*               csrSortedRowPtrA,
                        const int*               csrSortedColIndA,
                        mcDoubleComplex*         B,
                        int                      ldb,
                        csrsm2Info_t             info,
                        mcsparseSolvePolicy_t    policy,
                        void*                    pBuffer)
```

此函数执行稀疏三角线性系统 csrsm2 的求解阶段。参数 transB 作用于矩阵 B 和矩阵 X，只能取值为 MCSPARSE_OPERATION_NON_TRANSPOSE 和 MCSPARSE_OPERATION_TRANSPOSE。该操作是原地操作，即矩阵 B 被矩阵 X 覆盖。

如果 transB 为 MCSPARSE_OPERATION_NON_TRANSPOSE，则 ldb 必须不小于 m。否则，如果 transB 为 MCSPARSE_OPERATION_TRANSPOSE，则 ldb 必须不小于 nrhs。

此函数需要使用 csrsm2_bufferSize() 返回缓冲区大小。参数 pBuffer 的地址必须是 128 字节的倍数，否则将返回 MCSPARSE_STATUS_INVALID_VALUE。

尽管可以在不使用级别信息的情况下调用 csrsm2_solve()，但用户仍然需要注意一致性。如果在调用 csrsm2_analysis() 时使用了 MCSPARSE_SOLVE_POLICY_USE_LEVEL 策略，那么 csrsm2_solve() 可以在有或没有级别信息的情况下运行。相反，如果使用策略 MCSPARSE_SOLVE_POLICY_NO_LEVEL 调用了 csrsm2_analysis()，则 csrsm2_solve() 只能接受 MCSPARSE_SOLVE_POLICY_NO_LEVEL；否则，将返回 MCSPARSE_STATUS_INVALID_VALUE。

级别信息可能不会提高性能，但会花费额外的时间进行分析。例如，一个三对角矩阵没有并行性。在这种情况下，MCSPARSE_SOLVE_POLICY_NO_LEVEL 的性能优于 MCSPARSE_SOLVE_POLICY_USE_LEVEL。如果用户有一个迭代求解器，最佳方法是使用 MCSPARSE_SOLVE_POLICY_USE_LEVEL 进行一次 csrsm2_analysis()。然后，在第一次运行时使用 MCSPARSE_SOLVE_POLICY_NO_LEVEL 进行 csrsm2_solve()，在第二次运行时使用 MCSPARSE_SOLVE_POLICY_USE_LEVEL，选择速度更快的方式执行剩余的迭代。函数 csrsm2_solve() 报告第一个数值零的位置，包括结构零。如果 status 为 0，则没有找到数值零的位置。此外，如果指定了 MCSPARSE_DIAG_TYPE_UNIT，即使某些 j 的 A(j,j) 为零，也不报告数值零的位置。用户需要调用 mcsparseXcsrsm2_zeroPivot() 来获取数值零的位置。

csrsm2 提供了两种由参数 algo 指定的算法。algo=0 是非块版本，algo=1 是块版本。非块版本受内存带宽限制。块版本将矩阵划分为小的块，并应用稠密运算。尽管块版本的计算量比非块版本大，但如果非块版本已经达到最大带宽，块版本可能会更快。

如果 pBuffer != NULL，此函数具有以下特性。

- 该例程不需要额外的存储空间。
- 该例程支持异步执行。

输入

handle	处理 mcSPARSE 库上下文的句柄。
algo	algo = 0 是非阻塞版本。algo = 1 是阻塞版本。
transA	操作 op(A)。
transB	操作 op(B)。
m	矩阵 A 的行数。
nrhs	右侧矩阵 op(B) 的列数。
nnz	矩阵 A 的非零元素数。
alpha	用于乘法的 <type> 标量。
descrA	矩阵 A 的描述符。支持的矩阵类型为 MCSPARSE_MATRIX_TYPE_GENERAL，而支持的对角线类型为 MCSPARSE_DIAG_TYPE_UNIT 和 MCSPARSE_DIAG_TYPE_NON_UNIT。
csrValA	类型为 <type> 的数组，其中包含矩阵 A 中 nnz 个非零元素。
csrRowPtrA	由整数组成的长度为 m + 1 的数组，其中包含每一行的起始位置和最后一行结束位置加一。
csrColIndA	整数数组，长度为 nnz，包含矩阵 A 非零元素的列索引。
B	<type> 的右侧矩阵。op(B) 的大小为 m-by-nrhs
ldb	B 和 X 的主维度。
info	在分析阶段收集的信息的结构 (应该不经更改地传递到求解阶段)。
policy	支持的策略包括 MCSPARSE_SOLVE_POLICY_NO_LEVEL 和 MCSPARSE_SOLVE_POLICY_USE_LEVEL
pBuffer	用户分配的缓冲区，大小由 csrsm2_bufferSize() 返回。

输出

X	<type> 解矩阵，其中 op(X) 的大小为 m-by-nrhs。

有关返回状态的描述，请参见mcsparseStatus_t。

9.5 mcsparseXcsrsm2_zeroPivot()

```
mcsparseStatus_t
mcsparseXcsrsm2_zeroPivot(mcsparseHandle_t handle,
                          csrsm2Info_t     info,
                          int*             position)
```

如果返回的错误代码为 MCSPARSE_STATUS_ZERO_PIVOT，则 position=j 表示 A(j,j) 具有结构零元素或数值零元素。否则，position=-1。

position 可以是基于 0 或基于 1 的索引，与矩阵的索引方式相同。

函数 mcsparseXcsrsm2_zeroPivot() 是一个阻塞调用。它调用 mcDeviceSynchronize() 来确保所有之前的核操作已完成。

position 可以存在于主机内存或设备内存中。用户可以使用 mcsparseSetPointerMode() 来设置适当的模式。

- 该例程不需要额外的存储空间。
- 如果流式有序内存分配器可用，该例程支持异步执行。

输入

handle	处理 mcSPARSE 库上下文的句柄
info	如果用户已经调用了 csrsm2_analysis() 或 csrsm2_solve()，info 包含结构零或数值零

输出

position	如果没有结构或数值零，则 position 为 -1；否则，如果 A(j,j) 缺失或者 U(j,j) 为零，则 position=j。

有关返回状态的描述，请参见mcsparseStatus_t。

9.6 mcsparse<t>gemmi()

```
mcsparseStatus_t
mcsparseSgemmi(mcsparseHandle_t handle,
               int              m,
               int              n,
               int              k,
               int              nnz,
               const float*     alpha,
               const float*     A,
               int              lda,
               const float*     cscValB,
               const int*       cscColPtrB,
               const int*       cscRowIndB,
               const float*     beta,
               float*           C,
               int              ldc)

mcsparseStatus_t
mcsparseDgemmi(mcsparseHandle_t handle,
               int              m,
               int              n,
               int              k,
               int              nnz,
               const double*    alpha,
               const double*    A,
               int              lda,
               const double*    cscValB,
               const int*       cscColPtrB,
               const int*       cscRowIndB,
               const double*    beta,
               double*          C,
               int              ldc)

mcsparseStatus_t
mcsparseCgemmi(mcsparseHandle_t handle,
               int              m,
               int              n,
               int              k,
               int              nnz,
               const mcComplex* alpha,
               const mcComplex* A,
               int              lda,
               const mcComplex* cscValB,
               const int*       cscColPtrB,
               const int*       cscRowIndB,
               const mcComplex* beta,
```

(下页继续)

(续上页)

```
                mcComplex*        C,
                int               ldc)

mcsparseStatus_t
mcsparseZgemmi(mcsparseHandle_t       handle,
                int                   m,
                int                   n,
                int                   k,
                int                   nnz,
                const mcDoubleComplex* alpha,
                const mcDoubleComplex* A,
                int                   lda,
                const mcDoubleComplex* cscValB,
                const int*            cscColPtrB,
                const int*            cscRowIndB,
                const mcDoubleComplex* beta,
                mcDoubleComplex*      C,
                int                   ldc)
```

函数 `mcsparse<t>gemmi()` 执行以下矩阵乘法运算：

C=alpha*A*B+beta*c

其中，`A` 和 `C` 是稠密矩阵，`B` 是一个基于 CSC 存储格式的稀疏矩阵，由三个数组 `cscValB`、`cscColPtrB` 和 `cscRowIndB` 定义。`alpha` 和 `beta` 是标量值；注意：`B` 的索引是基于 0 的。

- 该例程不需要额外的存储空间。
- 该例程支持异步执行。

输入

`handle`	处理 mcSPARSE 库上下文的句柄。
`m`	矩阵 A 的行数。
`n`	矩阵 `B` 和 `C` 的列数。
`k`	矩阵 `A` 的列数。
`nnz`	矩阵 `B` 的非零元素个数。
`alpha`	用于乘法的 <type> 标量
`A`	维度为 `(lda, k)` 的数组。
`lda`	矩阵 `A` 的主维度，至少为 `m`
`cscValB`	维度为 `nnz` 的 <type> 数组，存储矩阵 `B` 的非零元素值。
`cscColPtrB`	长度为 `k+1` 的整数数组，包含每一行的起始位置和最后一行末尾位置加一。
`cscRowIndB`	维度为 `nnz` 的整数数组，存储矩阵 `B` 的非零元素列索引。
`beta`	用于乘法的 <type> 标量如果 beta 为 0，则 C 不需要是有效的输入。
`C`	维度为 `(ldc, n)` 的数组。
`ldc`	矩阵 `C` 的主维度，至少为 `m`

输出

`C`	更新后的维度为 `(ldc, n)` 的 <type> 数组。

有关返回状态的描述，请参见mcsparseStatus_t。

10 mcSPARSE 额外函数参考

本章介绍了用于操作稀疏矩阵的额外例程。

10.1 mcsparse<t>csrgeam2()

```
mcsparseStatus_t
mcsparseScsrgeam2_bufferSizeExt(mcsparseHandle_t         handle,
                                int                      m,
                                int                      n,
                                const float*             alpha,
                                const mcsparseMatDescr_t descrA,
                                int                      nnzA,
                                const float*             csrSortedValA,
                                const int*               csrSortedRowPtrA,
                                const int*               csrSortedColIndA,
                                const float*             beta,
                                const mcsparseMatDescr_t descrB,
                                int                      nnzB,
                                const float*             csrSortedValB,
                                const int*               csrSortedRowPtrB,
                                const int*               csrSortedColIndB,
                                const mcsparseMatDescr_t descrC,
                                const float*             csrSortedValC,
                                const int*               csrSortedRowPtrC,
                                const int*               csrSortedColIndC,
                                size_t*
 ↪pBufferSizeInBytes)

mcsparseStatus_t
mcsparseDcsrgeam2_bufferSizeExt(mcsparseHandle_t         handle,
                                int                      m,
                                int                      n,
                                const double*            alpha,
                                const mcsparseMatDescr_t descrA,
                                int                      nnzA,
                                const double*            csrSortedValA,
                                const int*               csrSortedRowPtrA,
                                const int*               csrSortedColIndA,
                                const double*            beta,
                                const mcsparseMatDescr_t descrB,
                                int                      nnzB,
                                const double*            csrSortedValB,
                                const int*               csrSortedRowPtrB,
```

(下页继续)

(续上页)

```
                                const int*                csrSortedColIndB,
                                const mcsparseMatDescr_t descrC,
                                const double*             csrSortedValC,
                                const int*                csrSortedRowPtrC,
                                const int*                csrSortedColIndC,
                                size_t*
 ↪pBufferSizeInBytes)

mcsparseStatus_t
mcsparseCcsrgeam2_bufferSizeExt(mcsparseHandle_t        handle,
                                int                     m,
                                int                     n,
                                const mcComplex*        alpha,
                                const mcsparseMatDescr_t descrA,
                                int                     nnzA,
                                const mcComplex*        csrSortedValA,
                                const int*              csrSortedRowPtrA,
                                const int*              csrSortedColIndA,
                                const mcComplex*        beta,
                                const mcsparseMatDescr_t descrB,
                                int                     nnzB,
                                const mcComplex*        csrSortedValB,
                                const int*              csrSortedRowPtrB,
                                const int*              csrSortedColIndB,
                                const mcsparseMatDescr_t descrC,
                                const mcComplex*        csrSortedValC,
                                const int*              csrSortedRowPtrC,
                                const int*              csrSortedColIndC,
                                size_t*
 ↪pBufferSizeInBytes)

mcsparseStatus_t
mcsparseZcsrgeam2_bufferSizeExt(mcsparseHandle_t         handle,
                                int                      m,
                                int                      n,
                                const mcDoubleComplex*   alpha,
                                const mcsparseMatDescr_t descrA,
                                int                      nnzA,
                                const mcDoubleComplex*   csrSortedValA,
                                const int*               csrSortedRowPtrA,
                                const int*               csrSortedColIndA,
                                const mcDoubleComplex*   beta,
                                const mcsparseMatDescr_t descrB,
                                int                      nnzB,
                                const mcDoubleComplex*   csrSortedValB,
                                const int*               csrSortedRowPtrB,
                                const int*               csrSortedColIndB,
                                const mcsparseMatDescr_t descrC,
                                const mcDoubleComplex*   csrSortedValC,
                                const int*               csrSortedRowPtrC,
                                const int*               csrSortedColIndC,
                                size_t*
 ↪pBufferSizeInBytes)

mcsparseStatus_t
mcsparseXcsrgeam2Nnz(mcsparseHandle_t         handle,
```

(下页继续)

(续上页)

```
                    int                         m,
                    int                         n,
                    const mcsparseMatDescr_t descrA,
                    int                         nnzA,
                    const int*                  csrSortedRowPtrA,
                    const int*                  csrSortedColIndA,
                    const mcsparseMatDescr_t descrB,
                    int                         nnzB,
                    const int*                  csrSortedRowPtrB,
                    const int*                  csrSortedColIndB,
                    const mcsparseMatDescr_t descrC,
                    int*                        csrSortedRowPtrC,
                    int*                        nnzTotalDevHostPtr,
                    void*                       workspace)
```

```
mcsparseStatus_t
mcsparseScsrgeam2(mcsparseHandle_t          handle,
                  int                       m,
                  int                       n,
                  const float*              alpha,
                  const mcsparseMatDescr_t descrA,
                  int                       nnzA,
                  const float*              csrSortedValA,
                  const int*                csrSortedRowPtrA,
                  const int*                csrSortedColIndA,
                  const float*              beta,
                  const mcsparseMatDescr_t descrB,
                  int                       nnzB,
                  const float*              csrSortedValB,
                  const int*                csrSortedRowPtrB,
                  const int*                csrSortedColIndB,
                  const mcsparseMatDescr_t descrC,
                  float*                    csrSortedValC,
                  int*                      csrSortedRowPtrC,
                  int*                      csrSortedColIndC,
                  void*                     pBuffer)

mcsparseStatus_t
mcsparseDcsrgeam2(mcsparseHandle_t          handle,
                  int                       m,
                  int                       n,
                  const double*             alpha,
                  const mcsparseMatDescr_t descrA,
                  int                       nnzA,
                  const double*             csrSortedValA,
                  const int*                csrSortedRowPtrA,
                  const int*                csrSortedColIndA,
                  const double*             beta,
                  const mcsparseMatDescr_t descrB,
                  int                       nnzB,
                  const double*             csrSortedValB,
                  const int*                csrSortedRowPtrB,
                  const int*                csrSortedColIndB,
                  const mcsparseMatDescr_t descrC,
                  double*                   csrSortedValC,
                  int*                      csrSortedRowPtrC,
```

(下页继续)

(续上页)

```
                  int*                       csrSortedColIndC,
                  void*                      pBuffer)

mcsparseStatus_t
mcsparseCcsrgeam2(mcsparseHandle_t           handle,
                  int                        m,
                  int                        n,
                  const mcComplex*           alpha,
                  const mcsparseMatDescr_t   descrA,
                  int                        nnzA,
                  const mcComplex*           csrSortedValA,
                  const int*                 csrSortedRowPtrA,
                  const int*                 csrSortedColIndA,
                  const mcComplex*           beta,
                  const mcsparseMatDescr_t   descrB,
                  int                        nnzB,
                  const mcComplex*           csrSortedValB,
                  const int*                 csrSortedRowPtrB,
                  const int*                 csrSortedColIndB,
                  const mcsparseMatDescr_t   descrC,
                  mcComplex*                 csrSortedValC,
                  int*                       csrSortedRowPtrC,
                  int*                       csrSortedColIndC,
                  void*                      pBuffer)

mcsparseStatus_t
mcsparseZcsrgeam2(mcsparseHandle_t           handle,
                  int                        m,
                  int                        n,
                  const mcDoubleComplex*     alpha,
                  const mcsparseMatDescr_t   descrA,
                  int                        nnzA,
                  const mcDoubleComplex*     csrSortedValA,
                  const int*                 csrSortedRowPtrA,
                  const int*                 csrSortedColIndA,
                  const mcDoubleComplex*     beta,
                  const mcsparseMatDescr_t   descrB,
                  int                        nnzB,
                  const mcDoubleComplex*     csrSortedValB,
                  const int*                 csrSortedRowPtrB,
                  const int*                 csrSortedColIndB,
                  const mcsparseMatDescr_t   descrC,
                  mcDoubleComplex*           csrSortedValC,
                  int*                       csrSortedRowPtrC,
                  int*                       csrSortedColIndC,
                  void*                      pBuffer)
```

通用程序如下：

```
int baseC, nnzC;
/* alpha, nnzTotalDevHostPtr 指向主机内存 */
size_t BufferSizeInBytes;
char *buffer = NULL;
int *nnzTotalDevHostPtr = &nnzC;
mcsparseSetPointerMode(handle, MCSPARSE_POINTER_MODE_HOST);
mcMalloc((void**)&csrRowPtrC, sizeof(int)*(m+1));
```

(下页继续)

(续上页)

```
/* 准备缓冲区 */
mcsparseScsrgeam2_bufferSizeExt(handle, m, n,
      alpha,
      descrA, nnzA,
      csrValA, csrRowPtrA, csrColIndA,
      beta,
      descrB, nnzB,
      csrValB, csrRowPtrB, csrColIndB,
      descrC,
      csrValC, csrRowPtrC, csrColIndC
      &bufferSizeInBytes
      );
mcMalloc((void**)&buffer, sizeof(char)*bufferSizeInBytes);
mcsparseXcsrgeam2Nnz(handle, m, n,
         descrA, nnzA, csrRowPtrA, csrColIndA,
         descrB, nnzB, csrRowPtrB, csrColIndB,
         descrC, csrRowPtrC, nnzTotalDevHostPtr,
         buffer);
if (NULL != nnzTotalDevHostPtr){
      nnzC = *nnzTotalDevHostPtr;
}else{
      mcMemcpy(&nnzC, csrRowPtrC+m, sizeof(int),
 ↪mcMemcpyDeviceToHost);
      mcMemcpy(&baseC, csrRowPtrC, sizeof(int),
 ↪mcMemcpyDeviceToHost);
      nnzC -= baseC;
}
mcMalloc((void**)&csrColIndC, sizeof(int)*nnzC);
mcMalloc((void**)&csrValC, sizeof(float)*nnzC);
mcsparseScsrgeam2(handle, m, n,
         alpha,
         descrA, nnzA,
         csrValA, csrRowPtrA, csrColIndA,
         beta,
         descrB, nnzB,
         csrValB, csrRowPtrB, csrColIndB,
         descrC,
         csrValC, csrRowPtrC, csrColIndC
         buffer);
```

对于 `csrgeam2()` 函数，有几点需要注意：

- 另外的三种组合：NT，TN 和 TT 不受 mcSPARSE 支持。如果要执行这三种组合之一，用户应使用例程 `csr2csc()` 进行转换。
- 仅支持 `MCSPARSE_MATRIX_TYPE_GENERAL` 类型的矩阵。如果 `A` 或 `B` 是对称或共轭转置的，则用户必须将矩阵扩展为完整矩阵，并将描述符的 `MatrixType` 字段重新配置为 `MCSPARSE_MATRIX_TYPE_GENERAL`。
- 如果已知矩阵 `C` 的稀疏模式，用户可以跳过调用 `mcsparseXcsrgeam2Nnz()` 函数。例如，假设用户有一个迭代算法，会迭代更新 `A` 和 `B`，但保持稀疏模式不变。用户可以调用 `mcsparseXcsrgeam2Nnz()` 函数一次来设置 `C` 的稀疏模式，然后仅在每次迭代中调用 `mcsparse[S|D|C|Z]geam()` 函数。
- 指针 `alpha` 和 `beta` 必须有效。
- 当 `alpha` 或 `beta` 为零时，并不被 `mcSPARSE` 视为特殊情况。`C` 的稀疏模式与 `alpha` 和 `beta` 的值无关。
- `csrgeam2()` 与 `csrgeam()` 类似，只是 `csrgeam2()` 需要显式缓冲区，而 `csrgeam()` 在内部分配缓冲区。

- 此函数需要内部分配的临时额外存储空间。
- 如果流式有序内存分配器可用，则该例程支持异步执行

输入

handle	处理 mcSPARSE 库上下文的句柄。
m	稀疏矩阵 A、B 和 C 的行数。
n	稀疏矩阵 A、B 和 C 的列数。
alpha	用于乘法的 <type> 标量。
descrA	矩阵 A 的描述符。支持的矩阵类型仅为 MCSPARSE_MATRIX_TYPE_GENERAL
nnzA	矩阵 A 的非零元素数量
csrValA	矩阵 A 的非零元素值数组，长度为 nnzA
csrRowPtrA	整型数组，包含 m+1 个元素，其中每个元素表示每一行的起始位置和最后一行的结束位置加一。
csrColIndA	整型数组，包含 nnzA 个元素，表示矩阵 A 非零元素的列索引。elements of matrix A。
beta	用于乘法操作的标量值。如果 beta 为零，则 y 不需要是一个有效输入。
descrB	矩阵 B 的描述符。支持的矩阵类型仅为 MCSPARSE_MATRIX_TYPE_GENERAL
nnzB	矩阵 B 的非零元素数量。
csrValB	矩阵 B 的非零元素值数组，长度为 nnzB。
csrRowPtrB	整型数组，包含 m+1 个元素，其中每个元素表示每一行的起始位置和最后一行的结束位置加一。
csrColIndB	整型数组，包含 nnzB 个元素，表示矩阵 B 非零元素的列索引。
descrC	矩阵 C 的描述符。支持的矩阵类型仅为 MCSPARSE_MATRIX_TYPE_GENERAL

输出

csrValC	矩阵 C 的非零元素值数组，长度为 nnzC
csrRowPtrC	整型数组，包含 m+1 个元素，其中每个元素表示每一行的起始位置和最后一行的结束位置加一。
csrColIndC	整型数组，包含 nnzC 个元素，表示矩阵 C 非零元素的列索引。
nnzTotalDevHostPtr	在设备或主机内存中非零元素的总数。它等于 (csrRowPtrC(m)-csrRowPtrC(0))

有关返回状态的描述，请参见mcsparseStatus_t。

10.2 mcsparse<t>csrgemm2()

```
mcsparseStatus_t
mcsparseScsrgemm2_bufferSizeExt(mcsparseHandle_t         handle,
                                int                      m,
                                int                      n,
                                int                      k,
                                const float*             alpha,
                                const mcsparseMatDescr_t descrA,
                                int                      nnzA,
                                const int*               csrRowPtrA,
                                const int*               csrColIndA,
                                const mcsparseMatDescr_t descrB,
                                int                      nnzB,
                                const int*               csrRowPtrB,
                                const int*               csrColIndB,
                                const float*             beta,
```

(下页继续)

(续上页)

```
                                const mcsparseMatDescr_t descrD,
                                int                      nnzD,
                                const int*               csrRowPtrD,
                                const int*               csrColIndD,
                                csrgemm2Info_t           info,
                                size_t*
 ↪pBufferSizeInBytes)

mcsparseStatus_t
mcsparseDcsrgemm2_bufferSizeExt(mcsparseHandle_t         handle,
                                int                      m,
                                int                      n,
                                int                      k,
                                const double*            alpha,
                                const mcsparseMatDescr_t descrA,
                                int                      nnzA,
                                const int*               csrRowPtrA,
                                const int*               csrColIndA,
                                const mcsparseMatDescr_t descrB,
                                int                      nnzB,
                                const int*               csrRowPtrB,
                                const int*               csrColIndB,
                                const double*            beta,
                                const mcsparseMatDescr_t descrD,
                                int                      nnzD,
                                const int*               csrRowPtrD,
                                const int*               csrColIndD,
                                csrgemm2Info_t           info,
                                size_t*
 ↪pBufferSizeInBytes)

mcsparseStatus_t
mcsparseCcsrgemm2_bufferSizeExt(mcsparseHandle_t         handle,
                                int                      m,
                                int                      n,
                                int                      k,
                                const mcComplex*         alpha,
                                const mcsparseMatDescr_t descrA,
                                int                      nnzA,
                                const int*               csrRowPtrA,
                                const int*               csrColIndA,
                                const mcsparseMatDescr_t descrB,
                                int                      nnzB,
                                const int*               csrRowPtrB,
                                const int*               csrColIndB,
                                const mcComplex*         beta,
                                const mcsparseMatDescr_t descrD,
                                int                      nnzD,
                                const int*               csrRowPtrD,
                                const int*               csrColIndD,
                                csrgemm2Info_t           info,
                                size_t*
 ↪pBufferSizeInBytes)

mcsparseStatus_t
mcsparseZcsrgemm2_bufferSizeExt(mcsparseHandle_t         handle,
```

(下页继续)

(续上页)

```
                                int                       m,
                                int                       n,
                                int                       k,
                                const mcDoubleComplex*    alpha,
                                const mcsparseMatDescr_t descrA,
                                int                       nnzA,
                                const int*                csrRowPtrA,
                                const int*                csrColIndA,
                                const mcsparseMatDescr_t descrB,
                                int                       nnzB,
                                const int*                csrRowPtrB,
                                const int*                csrColIndB,
                                const mcDoubleComplex*    beta,
                                const mcsparseMatDescr_t descrD,
                                int                       nnzD,
                                const int*                csrRowPtrD,
                                const int*                csrColIndD,
                                csrgemm2Info_t            info,
                                size_t*
 ↪pBufferSizeInBytes)

mcsparseStatus_t
mcsparseXcsrgemm2Nnz(mcsparseHandle_t         handle,
                     int                       m,
                     int                       n,
                     int                       k,
                     const mcsparseMatDescr_t descrA,
                     int                       nnzA,
                     const int*                csrRowPtrA,
                     const int*                csrColIndA,
                     const mcsparseMatDescr_t descrB,
                     int                       nnzB,
                     const int*                csrRowPtrB,
                     const int*                csrColIndB,
                     const mcsparseMatDescr_t descrD,
                     int                       nnzD,
                     const int*                csrRowPtrD,
                     const int*                csrColIndD,
                     const mcsparseMatDescr_t descrC,
                     int*                      csrRowPtrC,
                     int*                      nnzTotalDevHostPtr,
                     const csrgemm2Info_t      info,
                     void*                     pBuffer)
```

```
mcsparseStatus_t
mcsparseScsrgemm2(mcsparseHandle_t          handle,
                  int                       m,
                  int                       n,
                  int                       k,
                  const float*              alpha,
                  const mcsparseMatDescr_t descrA,
                  int                       nnzA,
                  const float*              csrValA,
                  const int*                csrRowPtrA,
                  const int*                csrColIndA,
                  const mcsparseMatDescr_t descrB,
```

(下页继续)

(续上页)

```
                  int                         nnzB,
                  const float*                csrValB,
                  const int*                  csrRowPtrB,
                  const int*                  csrColIndB,
                  const float*                beta,
                  const mcsparseMatDescr_t descrD,
                  int                         nnzD,
                  const float*                csrValD,
                  const int*                  csrRowPtrD,
                  const int*                  csrColIndD,
                  const mcsparseMatDescr_t descrC,
                  float*                      csrValC,
                  const int*                  csrRowPtrC,
                  int*                        csrColIndC,
                  const csrgemm2Info_t        info,
                  void*                       pBuffer)

mcsparseStatus_t
mcsparseDcsrgemm2(mcsparseHandle_t            handle,
                  int                         m,
                  int                         n,
                  int                         k,
                  const double*               alpha,
                  const mcsparseMatDescr_t descrA,
                  int                         nnzA,
                  const double*               csrValA,
                  const int*                  csrRowPtrA,
                  const int*                  csrColIndA,
                  const mcsparseMatDescr_t descrB,
                  int                         nnzB,
                  const double*               csrValB,
                  const int*                  csrRowPtrB,
                  const int*                  csrColIndB,
                  const double*               beta,
                  const mcsparseMatDescr_t descrD,
                  int                         nnzD,
                  const double*               csrValD,
                  const int*                  csrRowPtrD,
                  const int*                  csrColIndD,
                  const mcsparseMatDescr_t descrC,
                  double*                     csrValC,
                  const int*                  csrRowPtrC,
                  int*                        csrColIndC,
                  const csrgemm2Info_t        info,
                  void*                       pBuffer)

mcsparseStatus_t
mcsparseCcsrgemm2(mcsparseHandle_t            handle,
                  int                         m,
                  int                         n,
                  int                         k,
                  const mcComplex*            alpha,
                  const mcsparseMatDescr_t descrA,
                  int                         nnzA,
                  const mcComplex*            csrValA,
                  const int*                  csrRowPtrA,
```

(下页继续)

(续上页)

```
                  const int*                csrColIndA,
                  const mcsparseMatDescr_t  descrB,
                  int                       nnzB,
                  const mcComplex*          csrValB,
                  const int*                csrRowPtrB,
                  const int*                csrColIndB,
                  const mcComplex*          beta,
                  const mcsparseMatDescr_t  descrD,
                  int                       nnzD,
                  const mcComplex*          csrValD,
                  const int*                csrRowPtrD,
                  const int*                csrColIndD,
                  const mcsparseMatDescr_t  descrC,
                  mcComplex*                csrValC,
                  const int*                csrRowPtrC,
                  int*                      csrColIndC,
                  const csrgemm2Info_t      info,
                  void*                     pBuffer)

mcsparseStatus_t
mcsparseZcsrgemm2(mcsparseHandle_t          handle,
                  int                       m,
                  int                       n,
                  int                       k,
                  const mcDoubleComplex*    alpha,
                  const mcsparseMatDescr_t  descrA,
                  int                       nnzA,
                  const mcDoubleComplex*    csrValA,
                  const int*                csrRowPtrA,
                  const int*                csrColIndA,
                  const mcsparseMatDescr_t  descrB,
                  int                       nnzB,
                  const mcDoubleComplex*    csrValB,
                  const int*                csrRowPtrB,
                  const int*                csrColIndB,
                  const mcDoubleComplex*    beta,
                  const mcsparseMatDescr_t  descrD,
                  int                       nnzD,
                  const mcDoubleComplex*    csrValD,
                  const int*                csrRowPtrD,
                  const int*                csrColIndD,
                  const mcsparseMatDescr_t  descrC,
                  mcDoubleComplex*          csrValC,
                  const int*                csrRowPtrC,
                  int*                      csrColIndC,
                  const csrgemm2Info_t      info,
                  void*                     pBuffer)
```

此函数执行以下矩阵乘法操作：

C=alpha*A*B+beta*D

其中，A、B、D 和 C 分别为 m×k、k×n、m×n 和 m×n 的稀疏矩阵 (由数组 csrValA|csrValB|csrValD|csrValC 在 CSR 存储格式中定义)。

请注意，新的 API mcsparseSpGEMM 要求 D 必须具有与 C 相同的稀疏模式。

csrgemm2 使用 alpha 和 beta 来支持以下操作：

alpha	beta	operation
NULL	NULL	无效
NULL	!NULL	C = beta*D，不使用 A 和 B
!NULL	NULL	C = alpha*A*B，不使用 D
!NULL	!NULL	C = alpha*A*B + beta*D

alpha 和 beta 的数值只影响 C 的数值，而不影响其稀疏模式。例如，如果 alpha 和 beta 非零，则 C 的稀疏模式是 A*B 和 D 的并集，与 alpha 和 beta 的数值无关。

下表根据 m、n 和 k 的值显示了不同的操作：

m,n,k	操作
m<0 or n <0 or k<0	无效
m is 0 or n is 0	不执行任何操作
m >0 and n >0 and k is 0	若 beta 为零，则无效；若 beta 不为零，则 C = beta*D。
m >0 and n >0 and k >0	若 alpha 为零，则 C = beta*D；若 beta 为零，则 C = alpha*A*B；若 alpha 和 beta 都不为零，则 C = alpha*A*B + beta*D。

此函数需要由 csrgemm2_bufferSizeExt() 返回缓冲区大小。pBuffer 的地址必须是 128 字节的倍数。如果不是，将返回 MCSPARSE_STATUS_INVALID_VALUE。

mcSPARSE 库采用两步方法来完成稀疏矩阵计算。

在第一步中，用户分配具有 m+1 个元素的 csrRowPtrC，并使用函数 mcsparseXcsrgemm2Nnz() 来确定 csrRowPtrC 和非零元素的总数。在第二步中，用户从 (nnzC=*nnzTotalDevHostPtr) 或 (nnzC=csrRowPtrC(m)-csrRowPtrC(0)) 中获取 nnzC (矩阵 C 的非零元素数)，然后分别分配 nnzC 个元素的 csrValC 和 csrColIndC，最后调用函数 mcsparse[S|D|C|Z]csrgemm2() 对矩阵 C 进行求值。

C=-A*B+D 的通用程序如下：

```
// 假设矩阵 A，B 和 D 已经准备好。
int baseC, nnzC;
csrgemm2Info_t info = NULL;
size_t bufferSize;
void *buffer = NULL;
//nnzTotalDevHostPtr 指向主机内存
int *nnzTotalDevHostPtr = &nnzC;
double alpha = -1.0;
double beta  =  1.0;
mcsparseSetPointerMode(handle, MCSPARSE_POINTER_MODE_HOST);

// 步骤 1：创建一个不透明的结构
mcsparseCreateCsrgemm2Info(&info);

// 步骤 2：为 csrgemm2Nnz 和 csrgemm2 分配缓冲区
mcsparseDcsrgemm2_bufferSizeExt(handle, m, n, k, &alpha,
      descrA, nnzA, csrRowPtrA, csrColIndA,
      descrB, nnzB, csrRowPtrB, csrColIndB,
      &beta,
      descrD, nnzD, csrRowPtrD, csrColIndD,
      info,
      &bufferSize);
mcMalloc(&buffer, bufferSize);
```

(下页继续)

(续上页)

```
// 步骤 3：计算 csrRowPtrC
mcMalloc((void**)&csrRowPtrC, sizeof(int)*(m+1));
mcsparseXcsrgemm2Nnz(handle, m, n, k,
         descrA, nnzA, csrRowPtrA, csrColIndA,
         descrB, nnzB, csrRowPtrB, csrColIndB,
         descrD, nnzD, csrRowPtrD, csrColIndD,
         descrC, csrRowPtrC, nnzTotalDevHostPtr,
         info, buffer );
if (NULL != nnzTotalDevHostPtr){
      nnzC = *nnzTotalDevHostPtr;
}else{
      mcMemcpy(&nnzC, csrRowPtrC+m, sizeof(int),
↪mcMemcpyDeviceToHost);
      mcMemcpy(&baseC, csrRowPtrC, sizeof(int),
↪mcMemcpyDeviceToHost);
      nnzC -= baseC;
}

// 步骤 4：完成 C 的稀疏模式和值。
mcMalloc((void**)&csrColIndC, sizeof(int)*nnzC);
mcMalloc((void**)&csrValC, sizeof(double)*nnzC);
// 注意：如果只需要稀疏模式，将 csrValC 设置为 null。
mcsparseDcsrgemm2(handle, m, n, k, &alpha,
         descrA, nnzA, csrValA, csrRowPtrA, csrColIndA,
         descrB, nnzB, csrValB, csrRowPtrB, csrColIndB,
         &beta,
         descrD, nnzD, csrValD, csrRowPtrD, csrColIndD,
         descrC, csrValC, csrRowPtrC, csrColIndC,
         info, buffer);

// 步骤 5：销毁不透明的结构
mcsparseDestroyCsrgemm2Info(info);
```

对于 csrgemm2() 函数，有几点注意事项：

- 仅支持 NN 版本。对于其他模式，用户必须明确地转置矩阵 A 或 B。
- 仅支持 MCSPARSE_MATRIX_TYPE_GENERAL。如果 A 或 B 是对称的或共轭的，用户必须将矩阵扩展为完整矩阵，并重新配置 MatrixType 字段描述符为 MCSPARSE_MATRIX_TYPE_GENERAL。
- 如果 csrValC 为零，则仅计算 C 的稀疏模式。

如果 pBuffer != NULL，则函数 mcsparseXcsrgeam2Nnz() 和 mcsparse<t>csrgeam2() 具有以下特性：

- 该例程不需要额外的存储空间。
- 该例程支持异步执行。

输入

handle	处理 mcSPARSE 库上下文的句柄。
m	稀疏矩阵 A 、D 和 C 的行数。
n	稀疏矩阵 B 、D 和 C 的列数。
k	稀疏矩阵 A 和 B 的列数或行数。
alpha	用于乘法的 <type> 标量。
descrA	矩阵 A 的描述符。仅支持 MCSPARSE_MATRIX_TYPE_GENERAL 类型。
nnzA	矩阵 A 的非零元素个数。
csrValA	矩阵 A 的非零元素值数组。
csrRowPtrA	由 m+1 个元素组成的整型数组，包含每行的起始位置和最后一行的结束位置加一。
csrColIndA	整型数组，包含 nnzA 个矩阵 A 的非零元素的列索引。
descrB	矩阵 B 的描述符。仅支持 MCSPARSE_MATRIX_TYPE_GENERAL 类型。
nnzB	矩阵 B 的非零元素个数。
csrValB	矩阵 B 的非零元素值数组。
csrRowPtrB	由 k+1 个元素组成的整型数组，包含每行的起始位置和最后一行的结束位置加一。
csrColIndB	整型数组包含矩阵 B 中 nnzB 个非零元素的列索引。
descrD	矩阵 D 的描述符。仅支持 MCSPARSE_MATRIX_TYPE_GENERAL 类型。
nnzD	矩阵 D 的非零元素个数。
csrValD	矩阵 D 的非零元素值数组。
csrRowPtrD	由 m+1 个元素组成的整型数组，包含每行的起始位置和最后一行的结束位置加一。
csrColIndD	整型数组包含矩阵 D 中 nnzD 个非零元素的列索引。
beta	用于乘法的 <type> 标量。
descrC	矩阵 C 的描述符。仅支持 MCSPARSE_MATRIX_TYPE_GENERAL
info	在 csrgemm2Nnz 和 csrgemm2 中使用的存储有关信息的结构体。
pBuffer	用户分配的缓冲区，其大小由 csrgemm2_bufferSizeExt 返回。

输出

csrValC	稀疏矩阵 C 的非零元素值数组，长度为 nnzC。
csrRowPtrC	由 m+1 个元素组成的整型数组，用于记录每行起始位置和最后一行结束位置加一的索引。
csrColIndC	整型数组，包含矩阵 C 中 nnzC 个非零元素的列索引。
pBufferSizeInBytes	在 csrgemm2Nnnz 和 csrgemm2 中使用的缓冲区大小，以字节为单位。
nnzTotalDevHostPtr	稀疏矩阵 C 在设备或主机内存中的总非零元素个数，等于 (csrRowPtrC(m)-csrRowPtrC(0))。

有关返回状态的描述，请参见mcsparseStatus_t。

11 mcSPARSE 预处理器参考

本章描述了实现不同预处理器的例程。

11.1 不完全 Cholesky 分解：0 级

该部分讨论了用于 ic0 的不同算法。

11.1.1 mcsparse<t>csric02_bufferSize()

```
mcsparseStatus_t
mcsparseScsric02_bufferSize(mcsparseHandle_t         handle,
                            int                      m,
                            int                      nnz,
                            const mcsparseMatDescr_t descrA,
                            float*                   csrValA,
                            const int*               csrRowPtrA,
                            const int*               csrColIndA,
                            csric02Info_t            info,
                            int*                     pBufferSizeInBytes)

mcsparseStatus_t
mcsparseDcsric02_bufferSize(mcsparseHandle_t         handle,
                            int                      m,
                            int                      nnz,
                            const mcsparseMatDescr_t descrA,
                            double*                  csrValA,
                            const int*               csrRowPtrA,
                            const int*               csrColIndA,
                            csric02Info_t            info,
                            int*                     pBufferSizeInBytes)

mcsparseStatus_t
mcsparseCcsric02_bufferSize(mcsparseHandle_t         handle,
                            int                      m,
                            int                      nnz,
                            const mcsparseMatDescr_t descrA,
                            mcComplex*               csrValA,
                            const int*               csrRowPtrA,
                            const int*               csrColIndA,
                            csric02Info_t            info,
                            int*                     pBufferSizeInBytes)
```

(下页继续)

(续上页)

```
mcsparseStatus_t
mcsparseZcsric02_bufferSize(mcsparseHandle_t         handle,
                            int                      m,
                            int                      nnz,
                            const mcsparseMatDescr_t descrA,
                            mcDoubleComplex*         csrValA,
                            const int*               csrRowPtrA,
                            const int*               csrColIndA,
                            csric02Info_t            info,
                            int*                     pBufferSizeInBytes)
```

此函数返回用于计算不完全 Cholesky 因式分解 (0 填充和无主元) 的缓冲区大小。

A 是一个 m×m 的稀疏矩阵，是由三个数组 csrValA、csrRowPtrA 和 csrColIndA 在 CSR 存储格式中定义的。

缓冲区大小取决于维度 m 和矩阵非零元素的数量 nnz。如果用户更改了矩阵，需要再次调用 csric02_bufferSize() 函数来获取正确的缓冲区大小；否则可能会导致段错误。

- 该例程不需要额外的存储空间
- 该例程支持异步执行

输入

handle	处理 mcSPARSE 库上下文的句柄。
m	矩阵 A 的行数和列数。
nnz	矩阵 A 的非零元素个数。
descrA	矩阵 A 的描述符。支持的矩阵类型是 MCSPARSE_MATRIX_TYPE_GENERAL。此外，支持的索引基为 MCSPARSE_INDEX_BASE_ZERO 和 MCSPARSE_INDEX_BASE_ONE。
csrValA	<type> 数组，包含矩阵 A 中 nnz 个非零元素。
csrRowPtrA	包含 m + 1 个元素的整数数组，其中存储了每行的起始位置和最后一行结束位置加一的值。
csrColIndA	整数数组，包含矩阵 A 中 nnz 个非零元素的列索引。

输出

info	根据不同算法记录内部状态。
pBufferSizeInBytes	csric02_analysis() 和 csric02() 使用的缓冲区的字节数。

有关返回状态的描述，请参见mcsparseStatus_t。

11.1.2 mcsparse<t>csric02_analysis()

```
mcsparseStatus_t
mcsparseScsric02_analysis(mcsparseHandle_t         handle,
                          int                      m,
                          int                      nnz,
                          const mcsparseMatDescr_t descrA,
                          const float*             csrValA,
                          const int*               csrRowPtrA,
                          const int*               csrColIndA,
                          csric02Info_t            info,
                          mcsparseSolvePolicy_t    policy,
                          void*                    pBuffer)
```

(下页继续)

(续上页)

```
mcsparseStatus_t
mcsparseDcsric02_analysis(mcsparseHandle_t         handle,
                          int                      m,
                          int                      nnz,
                          const mcsparseMatDescr_t descrA,
                          const double*            csrValA,
                          const int*               csrRowPtrA,
                          const int*               csrColIndA,
                          csric02Info_t            info,
                          mcsparseSolvePolicy_t    policy,
                          void*                    pBuffer)

mcsparseStatus_t
mcsparseCcsric02_analysis(mcsparseHandle_t         handle,
                          int                      m,
                          int                      nnz,
                          const mcsparseMatDescr_t descrA,
                          const mcComplex*         csrValA,
                          const int*               csrRowPtrA,
                          const int*               csrColIndA,
                          csric02Info_t            info,
                          mcsparseSolvePolicy_t    policy,
                          void*                    pBuffer)

mcsparseStatus_t
mcsparseZcsric02_analysis(mcsparseHandle_t         handle,
                          int                      m,
                          int                      nnz,
                          const mcsparseMatDescr_t descrA,
                          const mcDoubleComplex*   csrValA,
                          const int*               csrRowPtrA,
                          const int*               csrColIndA,
                          csric02Info_t            info,
                          mcsparseSolvePolicy_t    policy,
                          void*                    pBuffer)
```

此函数执行不完全 Cholesky 因式分解 (0 填充，无主元) 的分析阶段：此函数需要一个由 `csric02_bufferSize()` 返回的缓冲区大小。`pBuffer` 的地址必须是 128 字节的倍数。如果不是，则返回 `MCSPARSE_STATUS_INVALID_VALUE`。

函数 `csric02_analysis()` 报告一个结构零，并计算存储在不透明结构 info 中的级别信息。级别信息可以在不完全 Cholesky 因式分解过程中提高并行效率。然而，`csric02()` 函数可以在没有级别信息的情况下执行。为了禁用级别信息，用户必须将 `csric02_analysis()` 和 `csric02()` 的策略指定为 `MCSPARSE_SOLVE_POLICY_NO_LEVEL`。

即使策略是 `MCSPARSE_SOLVE_POLICY_NO_LEVEL`，`csric02_analysis()` 函数始终报告第一个结构零。用户需要调用 `mcsparseXcsric02_zeroPivot()` 函数来获取结构零的位置信息。

如果 csric02_analysis() 报告一个结构零，用户可自行决定是否调用 csric02()。在这种情况下，用户仍然可以调用 `csric02()` 函数，它将在与结构零相同的位置返回一个数值零。然而，结果是没有意义的。

- 此函数需要内部分配的临时额外存储空间。
- 如果流式有序内存分配器可用，则该例程支持异步执行

输入

handle	处理 mcSPARSE 库上下文的句柄
m	矩阵 A 的行数和列数。
nnz	矩阵 A 的非零元素个数。
descrA	矩阵 A 的描述符。支持的矩阵类型为 MCSPARSE_MATRIX_TYPE_GENERAL。此外，支持的索引基地址为 MCSPARSE_INDEX_BASE_ZERO 和 MCSPARSE_INDEX_BASE_ONE。
csrValA	<type> 数组，包含矩阵 A 中 nnz 个非零元素。
csrRowPtrA	包含 m + 1 个元素的整数数组，其中存储了每行的起始位置和最后一行结束位置加一的值。
csrColIndA	整数数组，包含矩阵 A 中 nnz 个非零元素的列索引。
info	用 mcsparseCreateCsric02Info() 函数初始化结构体。
policy	支持的策略是 MCSPARSE_SOLVE_POLICY_NO_LEVEL 和 MCSPARSE_SOLVE_POLICY_USE_LEVEL
pBuffer	用户分配的缓冲区：大小由 csric02_bufferSize() 函数返回。

输出

info	在 csric02_analysis() 和 csric02() 中使用的缓冲区的字节数。

有关返回状态的描述，请参见mcsparseStatus_t。

11.1.3 mcsparse<t>csric02()

```
mcsparseStatus_t
mcsparseScsric02(mcsparseHandle_t         handle,
                 int                      m,
                 int                      nnz,
                 const mcsparseMatDescr_t descrA,
                 float*                   csrValA_valM,
                 const int*               csrRowPtrA,
                 const int*               csrColIndA,
                 csric02Info_t            info,
                 mcsparseSolvePolicy_t    policy,
                 void*                    pBuffer)

mcsparseStatus_t
mcsparseDcsric02(mcsparseHandle_t         handle,
                 int                      m,
                 int                      nnz,
                 const mcsparseMatDescr_t descrA,
                 double*                  csrValA_valM,
                 const int*               csrRowPtrA,
                 const int*               csrColIndA,
                 csric02Info_t            info,
                 mcsparseSolvePolicy_t    policy,
                 void*                    pBuffer)

mcsparseStatus_t
mcsparseCcsric02(mcsparseHandle_t         handle,
                 int                      m,
                 int                      nnz,
                 const mcsparseMatDescr_t descrA,
                 mcComplex*               csrValA_valM,
                 const int*               csrRowPtrA,
```

(下页继续)

(续上页)

```
                  const int*                csrColIndA,
                  csric02Info_t             info,
                  mcsparseSolvePolicy_t     policy,
                  void*                     pBuffer)

mcsparseStatus_t
mcsparseZcsric02(mcsparseHandle_t          handle,
                  int                       m,
                  int                       nnz,
                  const mcsparseMatDescr_t  descrA,
                  mcDoubleComplex*          csrValA_valM,
                  const int*                csrRowPtrA,
                  const int*                csrColIndA,
                  csric02Info_t             info,
                  mcsparseSolvePolicy_t     policy,
                  void*                     pBuffer)
```

此函数执行不完全 cholesky 因式分解计算 (0 填充、无主元) 的求解阶段：此函数需要由 csric02_bufferSize() 函数返回的缓冲区大小。pBuffer 的地址必须是 128 字节的倍数。如果不是，则返回 MCSPARSE_STATUS_INVALID_VALUE。

虽然 csric02() 函数可以在没有级别信息的情况下进行，但用户仍需注意一致性。如果在调用 csric02_analysis() 时使用了策略 MCSPARSE_SOLVE_POLICY_USE_LEVEL，那么 csric02() 可以在有或没有级别信息的情况下运行。另一方面，如果使用 MCSPARSE_SOLVE_POLICY_NO_LEVEL 调用 csric02_analysis() 函数，则 csric02() 函数只能接受 MCSPARSE_SOLVE_POLICY_NO_LEVEL；否则，返回 MCSPARSE_STATUS_INVALID_VALUE。

函数 csric02() 报告第一个数值零，包括结构零。用户必须调用 mcsparseXcsric02_zeroPivot() 以知道数值零的位置。

函数 csric02() 只使用矩阵 A 的下三角部分进行因式分解。矩阵类型必须是 MCSPARSE_MATRIX_TYPE_GENERAL，填充模式和对角线类型将被忽略，严格意义上三角部分被忽略且永远不会被修改。至于 A 是否是厄米矩阵并不重要。即，从 csric02() 的角度看，A 是厄米矩阵，只提供了下三角部分。

注解: 在实际应用中，正定矩阵可能没有不完全 Cholesky 因式分解。

据我们所知，只有矩阵 M 可以保证存在不完全 Cholesky 因式分解。如果 csric02() 未能进行 Cholesky 因式分解并报告了数值零，则可能不存在不完全 Cholesky 因式分解。

例如，假设 A 是一个实数 m×m 的矩阵，以下代码求解了预条件系统 M * y = x，其中 M 是 Cholesky 因式分解 L 及其转置的乘积。

如果 pBuffer != NULL，此函数支持以下特性：

- 此函数需要内部分配的临时额外存储空间。
- 如果流式有序内存分配器可用，则该例程支持异步执行。

输入

handle	处理 mcSPARSE 库上下文的句柄。
m	矩阵 A 的行数和列数。
nnz	矩阵 A 的非零元素个数。
descrA	矩阵 A 的描述符。支持的矩阵类型为 MCSPARSE_MATRIX_TYPE_GENERAL。此外，支持的索引基地址为 MCSPARSE_INDEX_BASE_ZERO 和 MCSPARSE_INDEX_BASE_ONE。
csrValA_valM	<type> 数组，包含矩阵 A 中 nnz 个非零元素。
csrRowPtrA	包含 m + 1 个元素的整数数组，其中存储了每行的起始位置和最后一行结束位置加一的值。
csrColIndA	整数数组，包含矩阵 A 中 nnz 个非零元素的列索引。
info	结构与分析阶段收集的信息相对应 (这些信息应该在求解阶段中保持不变)。
policy	支持的策略是 MCSPARSE_SOLVE_POLICY_NO_LEVEL 和 MCSPARSE_SOLVE_POLICY_USE_LEVEL
pBuffer	用户分配的缓冲区：大小由 csric02_bufferSize() 函数返回。

输出

csrValA_valM	包含不完全 Cholesky 下三角因式的 <type> 矩阵。

有关返回状态的描述，请参见mcsparseStatus_t。

11.1.4 mcsparseXcsric02_zeroPivot()

```
mcsparseStatus_t
mcsparseXcsric02_zeroPivot(mcsparseHandle_t handle,
                           csric02Info_t    info,
                           int*             position)
```

如果返回的错误代码是 MCSPARSE_STATUS_ZERO_PIVOT，则 position=j 表示 A(j,j) 要么是结构零，要么是数值零：否则，position=-1。

position 可以是以 0 为基准或者以 1 为基准，和矩阵一样。

函数 mcsparseXcsric02_zeroPivot() 是一个阻塞调用。它调用 mcDeviceSynchronize() 函数确保所有之前的核都已完成。

position 可以在主机内存或设备内存中。用户可以使用 mcsparseSetPointerMode() 来设置适当的模式。

- 该例程不需要额外的存储空间
- 如果流式有序内存分配器可用，则该例程支持异步执行

输入

handle	处理 mcSPARSE 库上下文的句柄。
info	如果用户已经调用了 csric02_analysis() 或 csric02()，则 info 包含结构零或数值零。

输出

position	如果没有结构性或数值上的零，则 position 为-1；否则，如果 A(j,j) 缺失或 L(j,j) 为零，则 position=j。

有关返回状态的描述，请参见mcsparseStatus_t。

11.2 不完全 LU 分解：0 级

本节中讨论了 ilu0 的不同算法。

11.2.1 mcsparse<t>csrilu02_numericBoost()

```
mcsparseStatus_t
mcsparseScsrilu02_numericBoost(mcsparseHandle_t handle,
                               csrilu02Info_t   info,
                               int              enable_boost,
                               double*          tol,
                               float*           boost_val)

mcsparseStatus_t
mcsparseDcsrilu02_numericBoost(mcsparseHandle_t handle,
                               csrilu02Info_t   info,
                               int              enable_boost,
                               double*          tol,
                               double*          boost_val)

mcsparseStatus_t
mcsparseCcsrilu02_numericBoost(mcsparseHandle_t handle,
                               csrilu02Info_t   info,
                               int              enable_boost,
                               double*          tol,
                               mcComplex*       boost_val)

mcsparseStatus_t
mcsparseZcsrilu02_numericBoost(mcsparseHandle_t handle,
                               csrilu02Info_t   info,
                               int              enable_boost,
                               double*          tol,
                               mcDoubleComplex* boost_val)
```

用户可以使用 boost 值来替换不完全 LU 因式分解中的数值。`tol` 用于确定一个数值零，`boost_val` 用于替换数值零。如果 `tol >= fabs(A(j,j))`，则 `A(j,j)=boost_val`。

要启用 boost 值，在调用 `csrilu02()` 之前，用户必须将参数 `enable_boost` 设置为 1。要禁用 boost 值，用户可以再次调用带有参数 `enable_boost=0` 的 `csrilu02_numericBoost()` 函数。

如果 `enable_boost=0`，则忽略 `tol` 和 `boost_val`。

`tol` 和 `boost_val` 都可以在主机内存或设备内存中。用户可以使用 `mcsparseSetPointerMode()` 函数设置适当的模式。

- 该例程不需要额外的存储空间。
- 该例程支持异步执行。

输入

`handle`	处理 mcSPARSE 库上下文的句柄。
`info`	使用 `mcsparseCreateCsrilu02Info()` 进行初始化结构。
`enable_boost`	通过 `enable_boost=0` 禁用 boost 值；否则，启用 boost 值。
`tol`	用于确定数值零的容差。
`boost_val`	用于替换数值零的 boost 值。

有关返回状态的描述，请参见mcsparseStatus_t。

11.2.2 mcsparse<t>csrilu02_bufferSize()

```
mcsparseStatus_t
mcsparseScsrilu02_bufferSize(mcsparseHandle_t         handle,
                             int                      m,
                             int                      nnz,
                             const mcsparseMatDescr_t descrA,
                             float*                   csrValA,
                             const int*               csrRowPtrA,
                             const int*               csrColIndA,
                             csrilu02Info_t           info,
                             int*                     pBufferSizeInBytes)

mcsparseStatus_t
mcsparseDcsrilu02_bufferSize(mcsparseHandle_t         handle,
                             int                      m,
                             int                      nnz,
                             const mcsparseMatDescr_t descrA,
                             double*                  csrValA,
                             const int*               csrRowPtrA,
                             const int*               csrColIndA,
                             csrilu02Info_t           info,
                             int*                     pBufferSizeInBytes)

mcsparseStatus_t
mcsparseCcsrilu02_bufferSize(mcsparseHandle_t         handle,
                             int                      m,
                             int                      nnz,
                             const mcsparseMatDescr_t descrA,
                             mcComplex*               csrValA,
                             const int*               csrRowPtrA,
                             const int*               csrColIndA,
                             csrilu02Info_t           info,
                             int*                     pBufferSizeInBytes)

mcsparseStatus_t
mcsparseZcsrilu02_bufferSize(mcsparseHandle_t         handle,
                             int                      m,
                             int                      nnz,
                             const mcsparseMatDescr_t descrA,
                             mcDoubleComplex*         csrValA,
                             const int*               csrRowPtrA,
                             const int*               csrColIndA,
                             csrilu02Info_t           info,
                             int*                     pBufferSizeInBytes)
```

此函数返回用于计算不完全 LU 因式分解 (0 填充，无主元) 的缓冲区大小。A 是一个由三个数组 csrValA、csrRowPtrA 和 csrColIndA 在 CSR 存储格式中定义的 mxm 的稀疏矩阵。

缓冲区大小取决于维度 m 和矩阵的非零元素数 nnz。如果用户更改了矩阵，需要再次调用 csrilu02_bufferSize() 函数来获得正确的缓冲区大小；否则可能会导致分段错误。

- 该例程不需要额外的存储空间
- 该例程支持异步执行

输入

handle	处理 mcSPARSE 库上下文的句柄。
m	矩阵 A 的行数和列数。
nnz	矩阵 A 的非零元素个数。
descrA	矩阵 A 的描述符。支持的矩阵类型为 MCSPARSE_MATRIX_TYPE_GENERAL。此外，支持的索引基地址为 MCSPARSE_INDEX_BASE_ZERO 和 MCSPARSE_INDEX_BASE_ONE。
csrValA	<type> 数组，包含矩阵 A 中 nnz 个非零元素。
csrRowPtrA	包含 m + 1 个元素的整数数组，其中存储了每行的起始位置和最后一行结束位置加一的值。
csrColIndA	整数数组，包含矩阵 A 中 nnz 个非零元素的列索引。

输出

info	根据不同算法记录内部状态。
pBufferSizeInBytes	csric02_analysis() 和 csric02() 使用的缓冲区的字节数。

有关返回状态的描述，请参见mcsparseStatus_t。

11.2.3 mcsparse<t>csrilu02_analysis()

```
mcsparseStatus_t
mcsparseScsrilu02_analysis(mcsparseHandle_t         handle,
                           int                      m,
                           int                      nnz,
                           const mcsparseMatDescr_t descrA,
                           const float*             csrValA,
                           const int*               csrRowPtrA,
                           const int*               csrColIndA,
                           csrilu02Info_t           info,
                           mcsparseSolvePolicy_t    policy,
                           void*                    pBuffer)

mcsparseStatus_t
mcsparseDcsrilu02_analysis(mcsparseHandle_t         handle,
                           int                      m,
                           int                      nnz,
                           const mcsparseMatDescr_t descrA,
                           const double*            csrValA,
                           const int*               csrRowPtrA,
                           const int*               csrColIndA,
                           csrilu02Info_t           info,
                           mcsparseSolvePolicy_t    policy,
                           void*                    pBuffer)

mcsparseStatus_t
mcsparseCcsrilu02_analysis(mcsparseHandle_t         handle,
                           int                      m,
                           int                      nnz,
                           const mcsparseMatDescr_t descrA,
                           const mcComplex*         csrValA,
                           const int*               csrRowPtrA,
                           const int*               csrColIndA,
                           csrilu02Info_t           info,
                           mcsparseSolvePolicy_t    policy,
```

(下页继续)

(续上页)

```
                            void*                       pBuffer)

mcsparseStatus_t
mcsparseZcsrilu02_analysis(mcsparseHandle_t          handle,
                            int                       m,
                            int                       nnz,
                            const mcsparseMatDescr_t  descrA,
                            const mcDoubleComplex*    csrValA,
                            const int*                csrRowPtrA,
                            const int*                csrColIndA,
                            csrilu02Info_t            info,
                            mcsparseSolvePolicy_t     policy,
                            void*                     pBuffer)
```

此函数执行不完全 LU 分解 (0 填充，无主元) 的分析阶段。A 是一个由三个数组 csrValA、csrRowPtrA 和 csrColIndA 在 CSR 存储格式中定义的 mxm 的稀疏矩阵。

此函数需要由 csrilu02_bufferSize() 返回的缓冲区大小。pBuffer 的地址必须是 128 字节的倍数。如果不是，则返回 MCSPARSE_STATUS_INVALID_VALUE。

函数 csrilu02_analysis() 报告一个结构零，并计算存储在不透明结构体 info 中的级别信息。级别信息可以在不完全 LU 分解过程中提高并行效率。然而，csrilu02() 也可以在没有级别信息的情况下执行。要禁用级别信息，用户必须将 csrilu02() 的策略设置为 MCSPARSE_SOLVE_POLICY_NO_LEVEL。

如果 csrilu02_analysis() 报告一个结构零，用户可以选择是否调用 csrilu02()。在这种情况下，用户仍然可以调用 csrilu02()，它将在与结构零相同的位置返回一个数值零。然而，结果是没有意义的。

- 此函数需要内部分配的额外临时存储空间
- 如果流式有序内存分配器可用，则该例程支持异步执行

输入

handle	处理 mcSPARSE 库上下文的句柄。
m	矩阵 A 的行数和列数。
nnz	矩阵 A 的非零元素个数。
descrA	矩阵 A 的描述符。支持的矩阵类型为 MCSPARSE_MATRIX_TYPE_GENERAL。此外，支持的索引基地址为 MCSPARSE_INDEX_BASE_ZERO 和 MCSPARSE_INDEX_BASE_ONE。
csrValA	<type> 数组，包含矩阵 A 中 nnz 个非零元素。
csrRowPtrA	包含 m + 1 个元素的整数数组，其中存储了每行的起始位置和最后一行结束位置加一的值。
csrColIndA	整数数组，包含矩阵 A 中 nnz 个非零元素的列索引。
info	使用 mcsparseCreateCsrilu02Info() 函数初始化结构。
policy	支持的策略是 MCSPARSE_SOLVE_POLICY_NO_LEVEL 和 MCSPARSE_SOLVE_POLICY_USE_LEVEL
pBuffer	用户分配的缓冲区：大小由 csrilu02_bufferSize() 函数返回。

输出

info	在分析阶段收集信息的结构 (应该在求解阶段传递时保持不变)。

有关返回状态的描述，请参见mcsparseStatus_t。

11.2.4 mcsparse<t>csrilu02()

```
mcsparseStatus_t
mcsparseScsrilu02(mcsparseHandle_t         handle,
                  int                      m,
                  int                      nnz,
                  const mcsparseMatDescr_t descrA,
                  float*                   csrValA_valM,
                  const int*               csrRowPtrA,
                  const int*               csrColIndA,
                  csrilu02Info_t           info,
                  mcsparseSolvePolicy_t    policy,
                  void*                    pBuffer)

mcsparseStatus_t
mcsparseDcsrilu02(mcsparseHandle_t         handle,
                  int                      m,
                  int                      nnz,
                  const mcsparseMatDescr_t descrA,
                  double*                  csrValA_valM,
                  const int*               csrRowPtrA,
                  const int*               csrColIndA,
                  csrilu02Info_t           info,
                  mcsparseSolvePolicy_t    policy,
                  void*                    pBuffer)

mcsparseStatus_t
mcsparseCcsrilu02(mcsparseHandle_t         handle,
                  int                      m,
                  int                      nnz,
                  const mcsparseMatDescr_t descrA,
                  mcComplex*               csrValA_valM,
                  const int*               csrRowPtrA,
                  const int*               csrColIndA,
                  csrilu02Info_t           info,
                  mcsparseSolvePolicy_t    policy,
                  void*                    pBuffer)

mcsparseStatus_t
mcsparseZcsrilu02(mcsparseHandle_t         handle,
                  int                      m,
                  int                      nnz,
                  const mcsparseMatDescr_t descrA,
                  mcDoubleComplex*         csrValA_valM,
                  const int*               csrRowPtrA,
                  const int*               csrColIndA,
                  csrilu02Info_t           info,
                  mcsparseSolvePolicy_t    policy,
                  void*                    pBuffer)
```

此函数执行不完全 LU 因式分解 (0 填充，无主元) 的求解阶段。A 是一个由三个数组 csrValA_valM、csrRowPtrA 和 csrColIndA 在 CSR 存储格式中定义的 mxm 稀疏矩阵。

此函数需要由 csrilu02_bufferSize() 返回的缓冲区大小。pBuffer 的地址必须是 128 字节的倍数。如果不是，则返回 MCSPARSE_STATUS_INVALID_VALUE。

矩阵类型必须是 MCSPARSE_MATRIX_TYPE_GENERAL。填充模式和对角线类型被忽略。

虽然 csrilu02() 可以在没有级别信息的情况下进行，但用户仍然需要注意一致性。

如果在调用 csrilu02_analysis() 时使用了策略 MCSPARSE_SOLVE_POLICY_USE_LEVEL，那么 csrilu02() 可以在有或没有级别信息的情况下运行。另一方面，如果调用 csrilu02_analysis() 时使用了策略 MCSPARSE_SOLVE_POLICY_NO_LEVEL，那么 csrilu02() 只能接受 MCSPARSE_SOLVE_POLICY_NO_LEVEL；否则，返回 MCSPARSE_STATUS_INVALID_VALUE。

函数 csrilu02() 报告第一个数值零，包括结构零。用户必须调用 mcsparseXcsrilu02_zeroPivot() 来知道数值零的位置。

例如，假设 A 是一个实数 mxm 矩阵，以下代码求解了预处理系统 M*y = x，其中 M 是 LU 因子 L 和 U 的乘积。

```
// 假设 A 是 m x m 的稀疏矩阵，以 CSR 格式表示。
// 假设条件：
// - 句柄已经由 mcsparseCreate() 创建。
// - (d_csrRowPtr, d_csrColInd, d_csrVal) 是 A 在设备内存中的 CSR 表
示。
// - d_x 是右侧向量在设备内存中的表示。
// - d_y 是解向量在设备内存中的表示。
// - d_z 是中间结果在设备内存中的表示。

mcsparseMatDescr_t descr_M = 0;
mcsparseMatDescr_t descr_L = 0;
mcsparseMatDescr_t descr_U = 0;
csrilu02Info_t info_M  = 0;
csrsv2Info_t  info_L  = 0;
csrsv2Info_t  info_U  = 0;
int pBufferSize_M;
int pBufferSize_L;
int pBufferSize_U;
int pBufferSize;
void *pBuffer = 0;
int structural_zero;
int numerical_zero;
const double alpha = 1.;
const mcsparseSolvePolicy_t policy_M = MCSPARSE_SOLVE_POLICY_NO_
↪LEVEL;
const mcsparseSolvePolicy_t policy_L = MCSPARSE_SOLVE_POLICY_NO_
↪LEVEL;
const mcsparseSolvePolicy_t policy_U = MCSPARSE_SOLVE_POLICY_USE_
↪LEVEL;
const mcsparseOperation_t trans_L  = MCSPARSE_OPERATION_NON_
↪TRANSPOSE;
const mcsparseOperation_t trans_U  = MCSPARSE_OPERATION_NON_
↪TRANSPOSE;

// 步骤 1：创建一个描述符，其中包含
// - 矩阵 M 的基索引为 1
// - 矩阵 L 的基索引为 1
// - 矩阵 L 是下三角形
// - 矩阵 L 的对角线元素为 1
// - 矩阵 U 的基索引为 1
// - 矩阵 U 是上三角形
// - 矩阵 U 的对角线元素不为 1
mcsparseCreateMatDescr(&descr_M);
mcsparseSetMatIndexBase(descr_M, MCSPARSE_INDEX_BASE_ONE);
mcsparseSetMatType(descr_M, MCSPARSE_MATRIX_TYPE_GENERAL);

mcsparseCreateMatDescr(&descr_L);
```

(下页继续)

(续上页)

```
mcsparseSetMatIndexBase(descr_L, MCSPARSE_INDEX_BASE_ONE);
mcsparseSetMatType(descr_L, MCSPARSE_MATRIX_TYPE_GENERAL);
mcsparseSetMatFillMode(descr_L, MCSPARSE_FILL_MODE_LOWER);
mcsparseSetMatDiagType(descr_L, MCSPARSE_DIAG_TYPE_UNIT);

mcsparseCreateMatDescr(&descr_U);
mcsparseSetMatIndexBase(descr_U, MCSPARSE_INDEX_BASE_ONE);
mcsparseSetMatType(descr_U, MCSPARSE_MATRIX_TYPE_GENERAL);
mcsparseSetMatFillMode(descr_U, MCSPARSE_FILL_MODE_UPPER);
mcsparseSetMatDiagType(descr_U, MCSPARSE_DIAG_TYPE_NON_UNIT);

// 步骤 2: 创建一个空的 info 结构体
// 我们需要一个 info 结构体用于 csrilu02, 和两个 info 结构体用于 csrsv2
mcsparseCreateCsrilu02Info(&info_M);
mcsparseCreateCsrsv2Info(&info_L);
mcsparseCreateCsrsv2Info(&info_U);

// 步骤 3: 查询 csrilu02 和 csrsv2 中所使用的内存量，并分配缓冲区
mcsparseDcsrilu02_bufferSize(handle, m, nnz,
      descr_M, d_csrVal, d_csrRowPtr, d_csrColInd, info_M, &
↪pBufferSize_M);
mcsparseDcsrsv2_bufferSize(handle, trans_L, m, nnz,
      descr_L, d_csrVal, d_csrRowPtr, d_csrColInd, info_L, &
↪pBufferSize_L);
mcsparseDcsrsv2_bufferSize(handle, trans_U, m, nnz,
      descr_U, d_csrVal, d_csrRowPtr, d_csrColInd, info_U, &
↪pBufferSize_U);

pBufferSize = max(pBufferSize_M, max(pBufferSize_L, pBufferSize_
↪U));

// mcMalloc 返回的 pBuffer 自动对齐到 128 字节。
mcMalloc((void**)&pBuffer, pBufferSize);

// 步骤 4: 对 M 执行不完全 Cholesky 分析
//         对 L 执行三角求解分析
//         对 U 执行三角求解分析
// M 的下 (上) 三角部分与 L(U) 具有相同的稀疏模式，
// 所以我们可以同时进行 csrilu0 和 csrsv2 的分析。

mcsparseDcsrilu02_analysis(handle, m, nnz, descr_M,
      d_csrVal, d_csrRowPtr, d_csrColInd, info_M,
      policy_M, pBuffer);
status = mcsparseXcsrilu02_zeroPivot(handle, info_M, &structural_
↪zero);
if (MCSPARSE_STATUS_ZERO_PIVOT == status){
   printf("A(%d,%d) is missing\n", structural_zero, structural_
↪zero);
}

mcsparseDcsrsv2_analysis(handle, trans_L, m, nnz, descr_L,
      d_csrVal, d_csrRowPtr, d_csrColInd,
      info_L, policy_L, pBuffer);

mcsparseDcsrsv2_analysis(handle, trans_U, m, nnz, descr_U,
      d_csrVal, d_csrRowPtr, d_csrColInd,
```

(下页继续)

(续上页)

```
      info_U, policy_U, pBuffer);

// 步骤 5:M = L * U
mcsparseDcsrilu02(handle, m, nnz, descr_M,
      d_csrVal, d_csrRowPtr, d_csrColInd, info_M, policy_M,
↪pBuffer);
status = mcsparseXcsrilu02_zeroPivot(handle, info_M, &numerical_
↪zero);
if (MCSPARSE_STATUS_ZERO_PIVOT == status){
   printf("U(%d,%d) is zero\n", numerical_zero, numerical_zero);
}

// 步骤 6: 解 L*z = x
mcsparseDcsrsv2_solve(handle, trans_L, m, nnz, &alpha, descr_L,
   d_csrVal, d_csrRowPtr, d_csrColInd, info_L,
   d_x, d_z, policy_L, pBuffer);

// 步骤 7: 解 U*y = z
mcsparseDcsrsv2_solve(handle, trans_U, m, nnz, &alpha, descr_U,
   d_csrVal, d_csrRowPtr, d_csrColInd, info_U,
   d_z, d_y, policy_U, pBuffer);

// 步骤 8: 释放资源
mcFree(pBuffer);
mcsparseDestroyMatDescr(descr_M);
mcsparseDestroyMatDescr(descr_L);
mcsparseDestroyMatDescr(descr_U);
mcsparseDestroyCsrilu02Info(info_M);
mcsparseDestroyCsrsv2Info(info_L);
mcsparseDestroyCsrsv2Info(info_U);
mcsparseDestroy(handle);
```

如果 `pBuffer != NULL`，此函数支持以下特性：

- 该例程不需要额外的存储空间
- 该例程支持异步执行

输入

handle	处理 mcSPARSE 库上下文的句柄。
m	矩阵 A 的行数和列数。
nnz	矩阵 A 的非零元素个数。
descrA	矩阵 A 的描述符。支持的矩阵类型为 MCSPARSE_MATRIX_TYPE_GENERAL。此外，支持的索引基地址为 MCSPARSE_INDEX_BASE_ZERO 和 MCSPARSE_INDEX_BASE_ONE。
csrValA_valM	<type> 数组，包含矩阵 A 中 nnz 个非零元素。
csrRowPtrA	包含 m + 1 个元素的整数数组，其中存储了每行的起始位置和最后一行结束位置加一的值。
csrColIndA	整数数组，包含矩阵 A 中 nnz 个非零元素的列索引。
info	结构与分析阶段收集的信息相对应 (这些信息应该在求解阶段中保持不变)。
policy	支持的策略是 MCSPARSE_SOLVE_POLICY_NO_LEVEL 和 MCSPARSE_SOLVE_POLICY_USE_LEVEL
pBuffer	用户分配的缓冲区：大小由 csrilu02_bufferSize() 函数返回。

输出

csrValA_valM	<type> 矩阵，包含不完全 LU 分解的下三角和上三角因子

有关返回状态的描述，请参见mcsparseStatus_t。

11.2.5 mcsparseXcsrilu02_zeroPivot()

```
mcsparseStatus_t
mcsparseXcsrilu02_zeroPivot(mcsparseHandle_t handle,
                            csrilu02Info_t   info,
                            int*             position)
```

如果返回的错误代码是 MCSPARSE_STATUS_ZERO_PIVOT，则 position=j 表示 A(j,j) 存在结构零或数值零；否则，position=-1。

position 可以是基于 0 或 1 的索引，与矩阵相同。

函数 mcsparseXcsrilu02_zeroPivot() 是一个阻塞调用。它调用 mcDeviceSynchronize() 确保所有先前的核已完成。

position 可以在主机内存或设备内存中。用户可以使用 mcsparseSetPointerMode() 设置适当的模式。

- 该例程不需要额外的存储空间
- 如果流式有序内存分配器可用，则该例程支持异步执行

输入

handle	处理 mcSPARSE 库上下文的句柄。
info	如果用户已经调用了 csrilu02_analysis() 或 csrilu02()，则 info 包含结构零或数值零。

输出

position	如果没有结构零或数值零，则 position 为-1; 否则，如果 A(j,j) 缺失或 U(j,j) 为零，则 position=j。

有关返回状态的描述，请参见mcsparseStatus_t。

11.3 三对角求解

三对角求解的不同求解算法在本部分中进行了讨论。

11.3.1 mcsparse<t>gtsv2_buffSizeExt()

```
mcsparseStatus_t
mcsparseSgtsv2_bufferSizeExt(mcsparseHandle_t handle,
                             int              m,
                             int              n,
                             const float*     dl,
                             const float*     d,
                             const float*     du,
                             const float*     B,
```

(下页继续)

(续上页)

```
                                int                 ldb,
                                size_t*             bufferSizeInBytes)

mcsparseStatus_t
mcsparseDgtsv2_bufferSizeExt(mcsparseHandle_t handle,
                                int                 m,
                                int                 n,
                                const double*       dl,
                                const double*       d,
                                const double*       du,
                                const double*       B,
                                int                 ldb,
                                size_t*             bufferSizeInBytes)

mcsparseStatus_t
mcsparseCgtsv2_bufferSizeExt(mcsparseHandle_t handle,
                                int                 m,
                                int                 n,
                                const mcComplex*    dl,
                                const mcComplex*    d,
                                const mcComplex*    du,
                                const mcComplex*    B,
                                int                 ldb,
                                size_t*             bufferSizeInBytes)

mcsparseStatus_t
mcsparseZgtsv2_bufferSizeExt(mcsparseHandle_t       handle,
                                int                 m,
                                int                 n,
                                const mcDoubleComplex*  dl,
                                const mcDoubleComplex*  d,
                                const mcDoubleComplex*  du,
                                const mcDoubleComplex*  B,
                                int                 ldb,
                                size_t*             bufferSizeInBytes)
```

此函数返回在计算具有多个右侧向量的三对角线性系统解的 `gtsv2` 中使用的缓冲区大小。

A*X=B

每个三对角线性系统的系数矩阵 `A` 由三个向量定义，分别对应于它的下对角线 (`dl`)、主对角线 (`d`) 和上对角线 (`du`)；右侧向量存储在稠密矩阵 `B` 中。请注意，在退出时，解 `X` 会覆盖右侧矩阵 `B`。

- 该例程不需要额外的存储空间
- 该例程支持异步执行

输入

handle	处理 mcSPARSE 库上下文的句柄。
m	线性系统的大小 (必须 >= 3)。
n	右侧向量的数量。矩阵 B 的列数。
dl	<type> 稠密数组，包含三对角线性系统的下对角线。每个下对角线的第一个元素必须为零。
d	<type> 稠密数组，包含三对角线性系统的主对角线。
du	<type> 稠密数组，包含三对角线性系统的上对角线。每个上对角线的最后一个元素必须为零。
B	<type> 稠密右侧向量，维度为 (ldb, n)。
ldb	B 的主维度。

输出

pBufferSizeInBytes	gtsv2 中使用的缓冲区的字节数。

有关返回状态的描述，请参见mcsparseStatus_t。

11.3.2 mcsparse<t>gtsv2()

```
mcsparseStatus_t
mcsparseSgtsv2(mcsparseHandle_t handle,
               int              m,
               int              n,
               const float*     dl,
               const float*     d,
               const float*     du,
               float*           B,
               int              ldb,
               void             pBuffer)

mcsparseStatus_t
mcsparseDgtsv2(mcsparseHandle_t handle,
               int              m,
               int              n,
               const double*    dl,
               const double*    d,
               const double*    du,
               double*          B,
               int              ldb,
               void             pBuffer)

mcsparseStatus_t
mcsparseCgtsv2(mcsparseHandle_t handle,
               int              m,
               int              n,
               const mcComplex* dl,
               const mcComplex* d,
               const mcComplex* du,
               mcComplex*       B,
               int              ldb,
               void             pBuffer)

mcsparseStatus_t
mcsparseZgtsv2(mcsparseHandle_t       handle,
```

(下页继续)

(续上页)

```
                int                       m,
                int                       n,
                const mcDoubleComplex*    dl,
                const mcDoubleComplex*    d,
                const mcDoubleComplex*    du,
                mcDoubleComplex*          B,
                int                       ldb,
                void                      pBuffer)
```

此函数用于计算具有多个右侧的三对角线性系统的解：

A*X=B

每个三对角线性系统的系数矩阵 A 定义为三个向量，分别对应于它的下对角线 (dl)、主对角线 (d) 和上对角线 (du)；右侧向量存储在稠密矩阵 B 中。请注意，在退出时，解 X 会覆盖的右侧向量矩阵 B。

假设 A 的大小为 m，以 1 为基准，dl、d 和 du 由以下公式定义：

- 对于 i=1,2,...,m，dl(i) := A(i, i-1)

dl 的第一个元素超出范围 (dl(1) := A(1,0))，因此 dl(1) = 0。

- 对于 i=1,2,...,m，d(i) = A(i,i)
- 对于 i=1,2,...,m，du(i) = A(i,i+1)

du 的最后一个元素超出范围 (du(m) := A(m,m+1))，因此 du(m) = 0。

该例程执行了主元选取，通常比 mcsparse<t>gtsv_nopivot() 或 mcsparse<t>gtsv2_nopivot() 产生更准确和稳定的结果，但需要一些执行时间。

此函数需要由 gtsv2_bufferSizeExt() 返回的缓冲区大小。pBuffer 的地址必须是 128 字节的倍数。如果不是，则返回 MCSPARSE_STATUS_INVALID_VALUE。

- 该例程不需要额外的存储空间
- 该例程支持异步执行

输入

handle	处理 mcSPARSE 库上下文的句柄。
m	线性系统的大小 (必须 >= 3)。
n	右侧向量的数量。矩阵 B 的列数。
dl	<type> 稠密数组，包含三对角线性系统的下对角线。每个下对角线的第一个元素必须为零。
d	<type> 稠密数组，包含三对角线性系统的主对角线。
du	<type> 稠密数组，包含三对角线性系统的上对角线。每个上对角线的最后一个元素必须为零。
B	<type> 稠密右侧向量，维度为 (ldb, n)。
ldb	B 的主维度。
pBuffer	用户分配的缓冲区，大小由 gtsv2_bufferSizeExt 返回。

输出

B	(ldb, n) 维度的 <type> 稠密解数组。

有关返回状态的描述，请参见mcsparseStatus_t。

11.3.3 mcsparse<t>gtsv2_nopivot_bufferSizeExt()

```
mcsparseStatus_t
mcsparseSgtsv2_nopivot_bufferSizeExt(mcsparseHandle_t handle,
                                     int              m,
                                     int              n,
                                     const float*     dl,
                                     const float*     d,
                                     const float*     du,
                                     const float*     B,
                                     int              ldb,
                                     size_t*          bufferSizeInBytes)

mcsparseStatus_t
mcsparseDgtsv2_nopivot_bufferSizeExt(mcsparseHandle_t handle,
                                     int              m,
                                     int              n,
                                     const double*    dl,
                                     const double*    d,
                                     const double*    du,
                                     const double*    B,
                                     int              ldb,
                                     size_t*          bufferSizeInBytes)

mcsparseStatus_t
mcsparseCgtsv2_nopivot_bufferSizeExt(mcsparseHandle_t handle,
                                     int              m,
                                     int              n,
                                     const mcComplex* dl,
                                     const mcComplex* d,
                                     const mcComplex* du,
                                     const mcComplex* B,
                                     int              ldb,
                                     size_t*          bufferSizeInBytes)

mcsparseStatus_t
mcsparseZgtsv2_nopivot_bufferSizeExt(mcsparseHandle_t       handle,
                                     int                    m,
                                     int                    n,
                                     const mcDoubleComplex* dl,
                                     const mcDoubleComplex* d,
                                     const mcDoubleComplex* du,
                                     const mcDoubleComplex* B,
                                     int                    ldb,
                                     size_t*
↪bufferSizeInBytes)
```

此函数返回 `gtsv2_nopivot` 中使用的缓冲区的大小，用于计算带有多个右侧的三对角线线性系统 (tri-diagonal linear system) 的解。

$$A*X=B$$

这些三对角线性系统的系数矩阵 `A` 由三个向量定义，分别对应于它的下对角线 (`dl`)、主对角线 (`d`) 和上对角线 (`du`)；右侧的值存储在稠密矩阵 `B` 中。请注意，在退出时，解 `X` 会覆盖右侧矩阵 `B`。

- 该例程不需要额外的存储空间。
- 该例程支持异步执行。

输入

handle	处理 mcSPARSE 库上下文的句柄。
m	线性系统的规模大小。(必须 >= 3)。
n	矩阵 B 的右侧标识数和列数。
dl	<type> 稠密数组，包含三对角线性系统的下对角线。每个下对角线的第一个元素必须为零。
d	<type> 稠密数组，包含三对角线性系统的主对角线。
du	<type> 稠密数组，包含三对角线性系统的上对角线。每个上对角线的最后一个元素必须为零。
B	一个维度为 (ldb, n) 的右侧 <type> 稠密数组。
ldb	矩阵 B 的主维度。

输出

pBufferSizeInBytes	缓冲区使用的字节数 gtsv2_nopivot。

有关返回状态的描述，请参见mcsparseStatus_t。

11.3.4 mcsparse<t>gtsv2_nopivot()

```
mcsparseStatus_t
mcsparseSgtsv2_nopivot(mcsparseHandle_t handle,
                       int              m,
                       int              n,
                       const float*     dl,
                       const float*     d,
                       const float*     du,
                       float*           B,
                       int              ldb,
                       void*            pBuffer)

mcsparseStatus_t
mcsparseDgtsv2_nopivot(mcsparseHandle_t handle,
                       int              m,
                       int              n,
                       const double*    dl,
                       const double*    d,
                       const double*    du,
                       double*          B,
                       int              ldb,
                       void*            pBuffer)

mcsparseStatus_t
mcsparseCgtsv2_nopivot(mcsparseHandle_t handle,
                       int              m,
                       int              n,
                       const mcComplex* dl,
                       const mcComplex* d,
                       const mcComplex* du,
                       mcComplex*       B,
                       int              ldb,
                       void*            pBuffer)
```

(下页继续)

(续上页)

```
mcsparseStatus_t
mcsparseZgtsv2_nopivot(mcsparseHandle_t       handle,
                       int                    m,
                       int                    n,
                       const mcDoubleComplex* dl,
                       const mcDoubleComplex* d,
                       const mcDoubleComplex* du,
                       mcDoubleComplex*       B,
                       int                    ldb,
                       void*                  pBuffer)
```

此函数用来计算多个右侧的三对角线线性系统的解：

A*X=B

每个三对角线性系统的系数矩阵 A 由三个向量定义，分别对应于它的下对角线 (dl)、主对角线 (d) 和上对角线 (du)；右侧的值存储在稠密矩阵 B 中。请注意，在退出时，解 X 会覆盖右侧矩阵 B。

该程序不执行任何主元素选取操作，而是使用循环归约 (Cyclic Reduction,CR) 和并行循环归约 (Parallel Cyclic Reduction,PCR) 这两种算法的组合来求解。当 m 是 2 次幂时，它可以实现更好的性能。

此函数需要一个由 gtsv2_nopivot_bufferSizeExt() 返回的缓冲区大小。pBuffer 的地址必须是 128 字节的倍数。如果不是，将返回 MCSPARSE_STATUS_INVALID_VALUE。

- 该例程不需要额外的存储空间。
- 该例程支持异步执行。

输入

handle	处理 mcSPARSE 库上下文的句柄。
m	线性系统的规模大小。(必须 >= 3)。
n	矩阵 B 的右侧标识数和列数。
dl	<type> 稠密数组，包含三对角线性系统的下对角线。每个下对角线的第一个元素必须为零。
d	<type> 稠密数组，包含三对角线性系统的主对角线。
du	<type> 稠密数组，包含三对角线性系统的上对角线。每个上对角线的最后一个元素必须为零。
B	一个维度为 (ldb, n) 的右侧 <type> 稠密数组。
ldb	矩阵 B 的主维度。
pBuffer	由用户分配的缓冲区，其大小由 gtsv2_nopivot_bufferSizeExt 决定。

输出

B	一个维度为 (ldb, n) 的 <type> 稠密解数组

有关返回状态的描述，请参见mcsparseStatus_t。

11.4 批处理三对角线求解 (Batched Tridiagonal Solve)

本节讨论了批处理三对角线线性求解的不同算法。

11.4.1 mcsparse<t>gtsv2StridedBatch_bufferSizeExt()

```
mcsparseStatus_t
mcsparseSgtsv2StridedBatch_bufferSizeExt(mcsparseHandle_t handle,
                                         int              m,
                                         const float*     dl,
                                         const float*     d,
                                         const float*     du,
                                         const float*     x,
                                         int              batchCount,
                                         int              batchStride,
                                         size_t*
 ↪bufferSizeInBytes)

mcsparseStatus_t
mcsparseDgtsv2StridedBatch_bufferSizeExt(mcsparseHandle_t handle,
                                         int              m,
                                         const double*    dl,
                                         const double*    d,
                                         const double*    du,
                                         const double*    x,
                                         int              batchCount,
                                         int              batchStride,
                                         size_t*
 ↪bufferSizeInBytes)

mcsparseStatus_t
mcsparseCgtsv2StridedBatch_bufferSizeExt(mcsparseHandle_t handle,
                                         int              m,
                                         const mcComplex* dl,
                                         const mcComplex* d,
                                         const mcComplex* du,
                                         const mcComplex* x,
                                         int              batchCount,
                                         int              batchStride,
                                         size_t*
 ↪bufferSizeInBytes)

mcsparseStatus_t
mcsparseZgtsv2StridedBatch_bufferSizeExt(mcsparseHandle_t       handle,
                                         int                    m,
                                         const mcDoubleComplex* dl,
                                         const mcDoubleComplex* d,
                                         const mcDoubleComplex* du,
                                         const mcDoubleComplex* x,
                                         int
 ↪batchCount,
                                         int
 ↪batchStride,
                                         size_t*
 ↪bufferSizeInBytes)
```

此函数返回用于 `gtsv2StridedBatch` 的缓冲区大小，此函数计算 i= 0,⋯,`batchCount` 之间的多个三对角线线性系统的解。

- 该例程不需要额外的存储空间。
- 该例程支持异步执行。

输入

handle	处理 mcSPARSE 库上下文的句柄。
n	线性系统的规模大小。(必须 >= 3)。
dl	<type> 稠密数组，包含三对角线性系统的下对角线。
d	<type> 稠密数组，包含三对角线性系统的主对角线。
du	<type> 稠密数组，包含三对角线性系统的上对角线。
x	一个包含三对角线性系统右侧项的 <type> 稠密数组。
batchCount	要求解的系统数量。
batchStride	将每个系统的向量分开的步幅 (元素的数量)(必须至少为 m)。

输出

pBufferSizeInBytes	在 gtsv2StridedBatch 中使用的缓冲区的字节数。

有关返回状态的描述，请参见mcsparseStatus_t。

11.4.2 mcsparse<t>gtsv2StridedBatch()

```
mcsparseStatus_t
mcsparseSgtsv2StridedBatch(mcsparseHandle_t handle,
                           int              m,
                           const float*     dl,
                           const float*     d,
                           const float*     du,
                           float*           x,
                           int              batchCount,
                           int              batchStride,
                           void*            pBuffer)

mcsparseStatus_t
mcsparseDgtsv2StridedBatch(mcsparseHandle_t handle,
                           int              m,
                           const double*    dl,
                           const double*    d,
                           const double*    du,
                           double*          x,
                           int              batchCount,
                           int              batchStride,
                           void*            pBuffer)

mcsparseStatus_t
mcsparseCgtsv2StridedBatch(mcsparseHandle_t handle,
                           int              m,
                           const mcComplex* dl,
                           const mcComplex* d,
                           const mcComplex* du,
                           mcComplex*       x,
                           int              batchCount,
                           int              batchStride,
                           void*            pBuffer)

mcsparseStatus_t
mcsparseZgtsv2StridedBatch(mcsparseHandle_t       handle,
```

(下页继续)

(续上页)

```
                                int                       m,
                                const mcDoubleComplex*    dl,
                                const mcDoubleComplex*    d,
                                const mcDoubleComplex*    du,
                                mcDoubleComplex*          x,
                                int                       batchCount,
                                int                       batchStride,
                                void*                     pBuffer)
```

此函数计算 i=0,…,batchCount 的多个三对角线线性系统的解。

每个三对角线线性系统的系数矩阵 A 是由其下对角线 (dl)、主对角线 (d) 和上对角线 (du) 对应的三个向量定义的；右侧的项存储在稠密矩阵 X 中。请注意，在退出时，解 Y 会覆盖右侧矩阵 X。假设不同的矩阵具有相同的大小，并且以固定的 batchStride 存储在内存中。

该程序不执行任何主元素选取操作，而是使用循环归约 (Cyclic Reduction,CR) 和并行循环归约 (Parallel Cyclic Reduction,PCR) 这两种算法的组合来求解。当 m 是 2 次幂时，它可以实现更好的性能。

此函数需要一个由 gtsv2_nopivot_bufferSizeExt() 返回的缓冲区大小。pBuffer 的地址必须是 128 字节的倍数。如果不是，将返回 MCSPARSE_STATUS_INVALID_VALUE。

- 该例程不需要额外的存储空间。
- 该例程支持异步执行。

输入

handle	处理 mcSPARSE 库上下文的句柄。
n	线性系统的规模大小。(必须 >= 3)。
dl	<type> 稠密数组，包含三对角线性系统的下对角线。
d	<type> 稠密数组，包含三对角线性系统的主对角线。
du	<type> 稠密数组，包含三对角线性系统的上对角线。
x	<type> 一个包含三对角线性系统右侧项的稠密数组。
batchCount	要求解的系统数量。
batchStride	将每个系统的向量分开的步幅 (元素的数量)(必须至少为 m)。
pBuffer	由用户分配的缓冲区，大小由 gtsv2StridedBatch_bufferSizeExt 返回。

输出

x	<type> 稠密数组，包含三对角线性系统的解。

有关返回状态的描述，请参见mcsparseStatus_t。

11.4.3 mcsparse<t>gtsvInterleavedBatch()

```
mcsparseStatus_t
mcsparseSgtsvInterleavedBatch_bufferSizeExt(mcsparseHandle_t handle,
                                            int              algo,
                                            int              m,
                                            const float*     dl,
                                            const float*     d,
                                            const float*     du,
                                            const float*     x,
                                            int              batchCount,
                                            size_t*          BufferSize)
```

(下页继续)

(续上页)

```
mcsparseStatus_t
mcsparseDgtsvInterleavedBatch_bufferSizeExt(mcsparseHandle_t handle,
                                            int              algo,
                                            int              m,
                                            const double*    dl,
                                            const double*    d,
                                            const double*    du,
                                            const double*    x,
                                            int              batchCount,
                                            size_t*          BufferSize)

mcsparseStatus_t
mcsparseCgtsvInterleavedBatch_bufferSizeExt(mcsparseHandle_t handle,
                                            int              algo,
                                            int              m,
                                            const mcComplex* dl,
                                            const mcComplex* d,
                                            const mcComplex* du,
                                            const mcComplex* x,
                                            int              batchCount,
                                            size_t*          BufferSize)

mcsparseStatus_t
mcsparseZgtsvInterleavedBatch_bufferSizeExt(mcsparseHandle_t       handle,
                                            int                    algo,
                                            int                    m,
                                            const mcDoubleComplex* dl,
                                            const mcDoubleComplex* d,
                                            const mcDoubleComplex* du,
                                            const mcDoubleComplex* x,
                                            int
 ↪batchCount,
                                            size_t*
 ↪BufferSize)
```

```
mcsparseStatus_t
mcsparseSgtsvInterleavedBatch(mcsparseHandle_t handle,
                              int              algo,
                              int              m,
                              float*           dl,
                              float*           d,
                              float*           du,
                              float*           x,
                              int              batchCount,
                              void*            pBuffer)

mcsparseStatus_t
mcsparseDgtsvInterleavedBatch(mcsparseHandle_t handle,
                              int              algo,
                              int              m,
                              double*          dl,
                              double*          d,
                              double*          du,
                              double*          x,
                              int              batchCount,
                              void*            pBuffer)
```

(下页继续)

(续上页)

```
mcsparseStatus_t
mcsparseCgtsvInterleavedBatch(mcsparseHandle_t handle,
                              int              algo,
                              int              m,
                              mcComplex*       dl,
                              mcComplex*       d,
                              mcComplex*       du,
                              mcComplex*       x,
                              int              batchCount,
                              void*            pBuffer)

mcsparseStatus_t
mcsparseZgtsvInterleavedBatch(mcsparseHandle_t handle,
                              int              algo,
                              int              m,
                              mcDoubleComplex* dl,
                              mcDoubleComplex* d,
                              mcDoubleComplex* du,
                              mcDoubleComplex* x,
                              int              batchCount,
                              void*            pBuffer)
```

此函数执行 csrsm2 的分析阶段，用于稀疏三角线性系统。

此函数计算 i=0,⋯,batchCount 的多个三对角线线性系统的解。

这些三对角线性系统的系数矩阵 A 由三个向量定义，分别对应于它的下对角线 (dl)、主对角线 (d) 和上对角线 (du)；右侧的值存储在稠密矩阵 B 中。请注意，在退出时，解 X 会覆盖右侧矩阵 B。

假设 A 的大小为 m 且基于 1 索引, dl, d 和 du 由以下公式定义：

dl(i) := A(i, i-1)，其中 i=1,2,...,m。

dl 的第一个元素超出边界 (dl(1) := A(1,0))，所以 dl(1) = 0。

- 对于 i=1,2,...,m 有 d(i) = A(i, i)。
- 对于 i=1,2,...,m 有 du(i) = A(i, i+1)。

du 的最后一个元素超出边界 (du(m) := A(m,m+1))，所以 du(m) = 0。

数据布局与 gtsvStridedBatch 不同，后者将所有矩阵按顺序聚合在一起。相反，gtsvInterleavedBatch 以连续的方式收集相同元素的不同矩阵。如果将 dl 视为大小为 m-by-batchCount 的二维数组，则 dl(:,j) 存储第 j 个矩阵。gtsvStridedBatch 使用列主序，而 gtsvInterleavedBatch 使用行主序。

该程序提供了三种不同的算法，由参数 algo 选择。第一种算法是由 Barcelona Supercomputing Center 提供的 mcThomas 算法。第二种算法是带有部分主元选取的 LU 分解算法，最后一种算法是 QR 分解算法。从稳定性的角度来看，mcThomas 算法在数值上不稳定，因为它没有进行主元选取。部分主元选取的 LU 分解算法和 QR 分解算法是稳定的。从性能的角度来看，带有部分主元选取的 LU 分解算法和 QR 分解算法比 mcThomas 算法慢大约 10% 到 20%。

此函数需要一个由 gtsvInterleavedBatch_bufferSizeExt() 返回的缓冲区大小。pBuffer 的地址必须是 128 字节的倍数。如果不是，将返回 MCSPARSE_STATUS_INVALID_VALUE。

如果用户准备了聚合格式，可以使用 mcblasXgeam 来获取交错格式。然而，这种转换所需要的时间与求解器本身相当。为了达到最佳性能，用户必须明确准备交错格式。

- 此函数需要内部分配的临时额外存储空间。
- 如果流式有序内存分配器可用，则该例程支持异步执行。

输入

handle	处理 mcSPARSE 库上下文的句柄。
algo	algo = 0: mcThomas(不稳定的算法); algo = 1: 带主元的 LU 分解算法 (稳定算法) algo = 2: QR 分解算法 (稳定算法)。
m	线性系统的大小
dl	<type> 稠密数组，包含三对角线性系统的下对角线。每个下对角线的第一个元素必须为零。
d	<type> 稠密数组，包含三对角线性系统的主对角线。
du	<type> 稠密数组，包含三对角线性系统的上对角线。每个上对角线的最后一个元素必须为零。
x	维度为 (batchCount, n) 的右侧 <type> 稠密数组。
pBuffer	由用户分配的缓冲区，大小由 gtsv2StridedBatch_bufferSizeExt 返回。

输出

x	维度为 (batchCount, n) 的 <type> 稠密解数组。

有关返回状态的描述，请参见mcsparseStatus_t。

11.5 批处理五对角线方程求解

本节讨论了批处理五对角线方程求解的不同算法。

11.5.1 mcsparse<t>gpsvInterleavedBatch()

```
mcsparseStatus_t
mcsparseSgpsvInterleavedBatch_bufferSizeExt(mcsparseHandle_t handle,
                                            int              algo,
                                            int              m,
                                            const float*     ds,
                                            const float*     dl,
                                            const float*     d,
                                            const float*     du,
                                            const float*     dw,
                                            const float*     x,
                                            int              batchCount,
                                            size_t*          BufferSize)

mcsparseStatus_t
mcsparseDgpsvInterleavedBatch_bufferSizeExt(mcsparseHandle_t handle,
                                            int              algo,
                                            int              m,
                                            const double*    ds,
                                            const double*    dl,
                                            const double*    d,
                                            const double*    du,
                                            const double*    dw,
                                            const double*    x,
                                            int              batchCount,
                                            size_t*          BufferSize)

mcsparseStatus_t
mcsparseCgpsvInterleavedBatch_bufferSizeExt(mcsparseHandle_t handle,
```

(下页继续)

(续上页)

```
                                          int                algo,
                                          int                m,
                                          const mcComplex*   ds,
                                          const mcComplex*   dl,
                                          const mcComplex*   d,
                                          const mcComplex*   du,
                                          const mcComplex*   dw,
                                          const mcComplex*   x,
                                          int                batchCount,
                                          size_t*            BufferSize)

mcsparseStatus_t
mcsparseZgpsvInterleavedBatch_bufferSizeExt(mcsparseHandle_t       handle,
                                          int                     algo,
                                          int                     m,
                                          const mcDoubleComplex*  ds,
                                          const mcDoubleComplex*  dl,
                                          const mcDoubleComplex*  d,
                                          const mcDoubleComplex*  du,
                                          const mcDoubleComplex*  dw,
                                          const mcDoubleComplex*  x,
                                          int
 ↪batchCount,
                                          size_t*
 ↪BufferSize)
```

```
mcsparseStatus_t
mcsparseSgpsvInterleavedBatch(mcsparseHandle_t handle,
                              int              algo,
                              int              m,
                              float*           ds,
                              float*           dl,
                              float*           d,
                              float*           du,
                              float*           dw,
                              float*           x,
                              int              batchCount,
                              void*            pBuffer)

mcsparseStatus_t
mcsparseDgpsvInterleavedBatch(mcsparseHandle_t handle,
                              int              algo,
                              int              m,
                              double*          ds,
                              double*          dl,
                              double*          d,
                              double*          du,
                              double*          dw,
                              double*          x,
                              int              batchCount,
                              void*            pBuffer)

mcsparseStatus_t
mcsparseCgpsvInterleavedBatch(mcsparseHandle_t handle,
                              int              algo,
                              int              m,
```

(下页继续)

(续上页)

```
                                mcComplex*        ds,
                                mcComplex*        dl,
                                mcComplex*        d,
                                mcComplex*        du,
                                mcComplex*        dw,
                                mcComplex*        x,
                                int               batchCount,
                                void*             pBuffer)

mcsparseStatus_t
mcsparseZgpsvInterleavedBatch(mcsparseHandle_t handle,
                                int               algo,
                                int               m,
                                mcDoubleComplex*  ds,
                                mcDoubleComplex*  dl,
                                mcDoubleComplex*  d,
                                mcDoubleComplex*  du,
                                mcDoubleComplex*  dw,
                                mcDoubleComplex*  x,
                                int               batchCount,
                                void*             pBuffer)
```

此函数计算多个五对角线性系统的解，对于 i=0,…,batchCount。

这些五对角线性系统的系数矩阵 A 由五个向量定义，分别对应于它的下对角线 (ds, dl)、主对角线 (d) 和上对角线 (du, dw)；右侧的值存储在稠密矩阵 B 中。请注意，解 X 在退出时会覆盖右侧矩阵 B。

假设 A 的大小为 m，从 1 开始计数，ds，dl，d，du 和 dw 由以下公式定义：

- 对于 i=1,2,...,m，ds(i) := A(i, i-2)

ds 的前两个元素超出边界 (ds(1) := A(1,-1)，ds(2) := A(2,0))，因此 ds(1) = 0 和 ds(2) = 0。

- 对于 i=1,2,...,m，dl(i) := A(i, i-1)

dl 的第一个元素超出边界 (dl(1) := A(1,0))，因此 dl(1) = 0 。

- 对于 i=1,2,...,m，d(i) = A(i,i)
- 对于 i=1,2,...,m，du(i) = A(i,i+1)

du 的最后一个元素超出边界 (du(m) := A(m,m+1))，因此 du(m) = 0 。

- 对于 i=1,2,...,m，dw(i) = A(i,i+2)

dw 的最后两个元素超出边界 (dw(m-1) := A(m-1,m+1)，dw(m) := A(m,m+2))，因此 dw(m-1) = 0 和 dw(m) = 0。

数据布局与 gtsvStridedBatch 相同。

该程序在数值上是稳定的，因为它使用 QR 分解来求解线性系统。

此函数需要一个由 gtsvInterleavedBatch_bufferSizeExt() 返回的缓冲区大小。pBuffer 的地址必须是 128 字节的倍数。如果不是，将返回 MCSPARSE_STATUS_INVALID_VALUE。

如果用户准备了聚合格式，可以使用 mcblasXgeam 来获取交错格式。然而，这种转换所需要的时间与求解器本身相当。为了达到最佳性能，用户必须明确准备交错格式。

如果 pBuffer != NULL，此函数支持以下特性：

- 该例程不需要额外的存储空间。
- 该例程支持异步执行。

输入

handle	处理 mcSPARSE 库上下文的句柄。
algo	仅支持 algo = 0(QR)。
m	线性系统的大小
ds	<type> 稠密数组，包含五对角线线性系统的下对角线 (距离主对角线为 2 的元素)。第一个元素和第二个元素必须为零。
dl	<type> 稠密数组，包含五对角线线性系统的下对角线 (距离主对角线为 1 的元素)。第一个元素必须为零。
d	<type> 稠密数组，包含五对角线线性系统的主对角线。
du	<type> 稠密数组，包含五对角线线性系统的上对角线 (距离主对角线为 1 的元素)。最后一个元素必须为零。
dw	<type> 稠密数组，包含五对角线线性系统的上对角线 (距离主对角线为 2 的元素)。最后两个元素必须为零。
x	维度为 (batchCount, n) 的的右侧 <type> 稠密数组。
pBuffer	由用户分配的缓冲区，大小由 gtsv2StridedBatch_bufferSizeExt 返回。

输出

x	维度为 (batchCount, n) 的 <type> 稠密解数组。

有关返回状态的描述，请参见mcsparseStatus_t。

12 mcSPARSE 重新排序参考

12.1 mcsparse<t>csrcolor()

```
mcsparseStatus_t
mcsparseScsrcolor(mcsparseHandle_t         handle,
                  int                      m,
                  int                      nnz,
                  const mcsparseMatDescr_t descrA,
                  const float*             csrValA,
                  const int*               csrRowPtrA,
                  const int*               csrColIndA,
                  const float*             fractionToColor,
                  int*                     ncolors,
                  int*                     coloring,
                  int*                     reordering,
                  mcsparseColorInfo_t      info)

mcsparseStatus_t
mcsparseDcsrcolor(mcsparseHandle_t         handle,
                  int                      m,
                  int                      nnz,
                  const mcsparseMatDescr_t descrA,
                  const double*            csrValA,
                  const int*               csrRowPtrA,
                  const int*               csrColIndA,
                  const double*            fractionToColor,
                  int*                     ncolors,
                  int*                     coloring,
                  int*                     reordering,
                  mcsparseColorInfo_t      info)

mcsparseStatus_t
mcsparseCcsrcolor(mcsparseHandle_t         handle,
                  int                      m,
                  int                      nnz,
                  const mcsparseMatDescr_t descrA,
                  const mcComplex*         csrValA,
                  const int*               csrRowPtrA,
                  const int*               csrColIndA,
                  const mcComplex*         fractionToColor,
                  int*                     ncolors,
                  int*                     coloring,
                  int*                     reordering,
                  mcsparseColorInfo_t      info)
```

(下页继续)

(续上页)

```
mcsparseStatus_t
mcsparseZcsrcolor(mcsparseHandle_t          handle,
                  int                       m,
                  int                       nnz,
                  const mcsparseMatDescr_t  descrA,
                  const mcDoubleComplex*    csrValA,
                  const int*                csrRowPtrA,
                  const int*                csrColIndA,
                  const mcDoubleComplex*    fractionToColor,
                  int*                      ncolors,
                  int*                      coloring,
                  int*                      reordering,
                  mcsparseColorInfo_t       info)
```

此函数对 CSR 格式存储的矩阵 A 中的关联邻接图进行着色。着色是将颜色 (整数) 分配给节点的过程，使相邻的节点具有不同的颜色。在这个例程中，使用了近似着色算法，并在一定比例的节点被着色后停止。其余的节点被分配不同的颜色 (从之前使用的最后一个整数开始的递增整数序列)。最后两个辅助程序可用于提取得到的颜色数量、其分配和相关的重新排序。重新排序使得被分配相同颜色的节点相邻排列在一起。

传递给这个程序的矩阵 A 必须以一般矩阵的形式存储，并且具有对称的稀疏模式。如果矩阵是非对称的，用户应该将 A+A^T 作为参数传递给这个程序。

- 此函数需要内部分配的临时额外存储空间。
- 如果流式有序内存分配器可用，则该例程支持异步执行。

输入

handle	处理 mcSPARSE 库上下文的句柄。
m	矩阵 A 的行数。
nnz	矩阵 A 的非零元素个数。
descrA	矩阵 A 的描述符，支持的矩阵类型为 MCSPARSE_MATRIX_TYPE_GENERAL。此外，支持的索引基数为 MCSPARSE_INDEX_BASE_ZERO 和 MCSPARSE_INDEX_BASE_ONE。
csrValA	整数数组，包含矩阵 A 中 nnz 个非零元素的列索引。
csrRowPtrA	包含 m+1 个元素，其中每个元素表示每一行的起始位置和最后一行的结束位置加一。
csrColIndA	矩阵 A 的非零元素的 nnz 个列索引的整数数组。
fractionToColor	节点被着色的比例，取值范围为 [0.0, 1.0]，例如 0.8 表示将着色 80% 的节点。
info	包含传递给着色算法的信息的数据结构。

输出

ncolors	使用的不同颜色数量 (最多为矩阵的大小，但很可能比它小得多)。
coloring	得到的着色排列结果。
reordering	得到的重新排序排列 (如果为 NULL，则保持不变)。

有关返回状态的描述，请参见mcsparseStatus_t。

13 mcSPARSE 格式转换参考

本章介绍了不同稀疏和稠密存储格式之间的转换程序。

13.1 mcsparse<t>csr2gebsr()

```
mcsparseStatus_t
mcsparseScsr2gebsr_bufferSize(mcsparseHandle_t          handle,
                              mcsparseDirection_t       dir,
                              int                       m,
                              int                       n,
                              const mcsparseMatDescr_t  descrA,
                              const float*              csrValA,
                              const int*                csrRowPtrA,
                              const int*                csrColIndA,
                              int                       rowBlockDim,
                              int                       colBlockDim,
                              int*                      pBufferSize)

mcsparseStatus_t
mcsparseDcsr2gebsr_bufferSize(mcsparseHandle_t          handle,
                              mcsparseDirection_t       dir,
                              int                       m,
                              int                       n,
                              const mcsparseMatDescr_t  descrA,
                              const double*             csrValA,
                              const int*                csrRowPtrA,
                              const int*                csrColIndA,
                              int                       rowBlockDim,
                              int                       colBlockDim,
                              int*                      pBufferSize)

mcsparseStatus_t
mcsparseCcsr2gebsr_bufferSize(mcsparseHandle_t          handle,
                              mcsparseDirection_t       dir,
                              int                       m,
                              int                       n,
                              const mcsparseMatDescr_t  descrA,
                              const mcComplex*          csrValA,
                              const int*                csrRowPtrA,
                              const int*                csrColIndA,
                              int                       rowBlockDim,
                              int                       colBlockDim,
                              int*                      pBufferSize)
```

(下页继续)

(续上页)

```
mcsparseStatus_t
mcsparseZcsr2gebsr_bufferSize(mcsparseHandle_t         handle,
                              mcsparseDirection_t      dir,
                              int                      m,
                              int                      n,
                              const mcsparseMatDescr_t descrA,
                              const mcDoubleComplex*   csrValA,
                              const int*               csrRowPtrA,
                              const int*               csrColIndA,
                              int                      rowBlockDim,
                              int                      colBlockDim,
                              int*                     pBufferSize)
```

```
mcsparseStatus_t
mcsparseXcsr2gebsrNnz(mcsparseHandle_t      handle,
                      mcsparseDirection_t      dir,
                      int                      m,
                      int                      n,
                      const mcsparseMatDescr_t descrA,
                      const int*               csrRowPtrA,
                      const int*               csrColIndA,
                      const mcsparseMatDescr_t descrC,
                      int*                     bsrRowPtrC,
                      int                      rowBlockDim,
                      int                      colBlockDim,
                      int*                     nnzTotalDevHostPtr,
                      void*                    pBuffer)

mcsparseStatus_t
mcsparseScsr2gebsr(mcsparseHandle_t       handle,
                   mcsparseDirection_t      dir,
                   int                      m,
                   int                      n,
                   const mcsparseMatDescr_t descrA,
                   const float*             csrValA,
                   const int*               csrRowPtrA,
                   const int*               csrColIndA,
                   const mcsparseMatDescr_t descrC,
                   float*                   bsrValC,
                   int*                     bsrRowPtrC,
                   int*                     bsrColIndC,
                   int                      rowBlockDim,
                   int                      colBlockDim,
                   void*                    pBuffer)

mcsparseStatus_t
mcsparseDcsr2gebsr(mcsparseHandle_t       handle,
                   mcsparseDirection_t      dir,
                   int                      m,
                   int                      n,
                   const mcsparseMatDescr_t descrA,
                   const double*            csrValA,
                   const int*               csrRowPtrA,
                   const int*               csrColIndA,
                   const mcsparseMatDescr_t descrC,
```

(下页继续)

(续上页)

```
                    double*                 bsrValC,
                    int*                    bsrRowPtrC,
                    int*                    bsrColIndC,
                    int                     rowBlockDim,
                    int                     colBlockDim,
                    void*                   pBuffer)

mcsparseStatus_t
mcsparseCcsr2gebsr(mcsparseHandle_t         handle,
                    mcsparseDirection_t     dir,
                    int                     m,
                    int                     n,
                    const mcsparseMatDescr_t descrA,
                    const mcComplex*        csrValA,
                    const int*              csrRowPtrA,
                    const int*              csrColIndA,
                    const mcsparseMatDescr_t descrC,
                    mcComplex*              bsrValC,
                    int*                    bsrRowPtrC,
                    int*                    bsrColIndC,
                    int                     rowBlockDim,
                    int                     colBlockDim,
                    void*                   pBuffer)

mcsparseStatus_t
mcsparseZcsr2gebsr(mcsparseHandle_t         handle,
                    mcsparseDirection_t     dir,
                    int                     m,
                    int                     n,
                    const mcsparseMatDescr_t descrA,
                    const mcDoubleComplex*  csrValA,
                    const int*              csrRowPtrA,
                    const int*              csrColIndA,
                    const mcsparseMatDescr_t descrC,
                    mcDoubleComplex*        bsrValC,
                    int*                    bsrRowPtrC,
                    int*                    bsrColIndC,
                    int                     rowBlockDim,
                    int                     colBlockDim,
                    void*                   pBuffer)
```

此函数将以 CSR 格式 (由数组 csrValA、csrRowPtrA 和 csrColIndA 定义) 的稀疏矩阵 A 转换为以一般 BSR 格式定义的稀疏矩阵 C (由三个数组 bsrValC、bsrRowPtrC 和 bsrColIndC 定义)。

矩阵 A 是一个大小为 m*n 的稀疏矩阵，矩阵 C 是一个大小为 (mb*rowBlockDim)*(nb*colBlockDim) 的稀疏矩阵，其中 mb(=(m+rowBlockDim-1)/rowBlockDim) 是 C 的块行数，nb(=(n+colBlockDim-1)/colBlockDim) 是 C 的块列数。

矩阵 C 的块大小为 rowBlockDim*colBlockDim。如果 m 不是 rowBlockDim 的倍数或者 n 不是 colBlockDim 的倍数，将填充零值。

该实现通过以下操作进行转换：

首先，用户分配了大小为 mb+1 的 bsrRowPtrC 数组，并使用函数 mcsparseXcsr2gebsrNnz() 来确定每个块行中的非零块列数。其次，用户从 nnzb=*nnzTotalDevHostPtr 或者 nnzb=bsrRowPtrC[mb]-bsrRowPtrC[0] 中获取了 nnzb (矩阵 C 的非零块列数)，并分配了大小为 nnzb*rowBlockDim*colBlockDim 的 bsrValC 数组和大小为 nnzb 的整数数组 bsrColIndC。最后，调用函数 mcsparse[S|D|C|Z]csr2gebsr() 完成转换。

用户必须通过调用 csr2gebsr_bufferSize() 来获取 csr2gebsr() 所需的缓冲区大小，然后分配缓冲区，并将缓冲区指针传递给 csr2gebsr()。

一般程序如下所示：

```
// 给定 CSR 格式 (csrRowPtrA、csrColIndA、csrValA) 和
// BSR 格式的块以列优先顺序存储。
mcsparseDirection_t dir = MCSPARSE_DIRECTION_COLUMN;
int base, nnzb;
int mb = (m + rowBlockDim-1)/rowBlockDim;
int nb = (n + colBlockDim-1)/colBlockDim;
int bufferSize;
void *pBuffer;
mcsparseScsr2gebsr_bufferSize(handle, dir, m, n,
      descrA, csrValA, csrRowPtrA, csrColIndA,
      rowBlockDim, colBlockDim,
      &bufferSize);
mcMalloc((void**)&pBuffer, bufferSize);
mcMalloc((void**)&bsrRowPtrC, sizeof(int) *(mb+1));
// nnzTotalDevHostPtr 指向主机内存
int *nnzTotalDevHostPtr = &nnzb;
mcsparseXcsr2gebsrNnz(handle, dir, m, n,
      descrA, csrRowPtrA, csrColIndA,
      descrC, bsrRowPtrC, rowBlockDim, colBlockDim,
      nnzTotalDevHostPtr,
      pBuffer);
if (NULL != nnzTotalDevHostPtr){
      nnzb = *nnzTotalDevHostPtr;
}else{
      mcMemcpy(&nnzb, bsrRowPtrC+mb, sizeof(int),
 ↪mcMemcpyDeviceToHost);
      mcMemcpy(&base, bsrRowPtrC, sizeof(int),
 ↪mcMemcpyDeviceToHost);
      nnzb -= base;
}
mcMalloc((void**)&bsrColIndC, sizeof(int)*nnzb);
mcMalloc((void**)&bsrValC,
 ↪sizeof(float)*(rowBlockDim*colBlockDim)*nnzb);
mcsparseScsr2gebsr(handle, dir, m, n,
        descrA,
        csrValA, csrRowPtrA, csrColIndA,
        descrC,
        bsrValC, bsrRowPtrC, bsrColIndC,
        rowBlockDim, colBlockDim,
        pBuffer);
```

- 函数 mcsparseXcsr2gebsrNnz() 具有以下特点：
  - 此函数需要内部分配的临时额外存储空间。
  - 如果流式有序内存分配器可用，则该例程支持异步执行。
- 如果 pBuffer != NULL，函数 mcsparse<t>csr2gebsr() 具有以下特点：
  - 该程序支持异步执行。

输入

handle	处理 mcSPARSE 库上下文的句柄。
dir	块的存储格式，可以是 MCSPARSE_DIRECTION_ROW 或 MCSPARSE_DIRECTION_COLUMN
m	稀疏矩阵 A 的行数。
n	稀疏矩阵 A 的列数。
descrA	矩阵 A 的描述符。支持的矩阵类型为 MCSPARSE_MATRIX_TYPE_GENERAL。此外，支持的索引基数为 MCSPARSE_INDEX_BASE_ZERO 和 MCSPARSE_INDEX_BASE_ONE。
csrValA	<type> 矩阵 A 中 nnz 个非零元素的数组。
csrRowPtrA	包含 m+1 个元素，中每个元素表示每一行的起始位置和其最后一行的结束位置加一。
csrColIndA	整数数组，包含矩阵 A 中 nnz 个非零元素的列索引
descrC	矩阵 C 的描述符。支持的矩阵类型是 MCSPARSE_MATRIX_TYPE_GENERAL。此外，支持的索引基数为 MCSPARSE_INDEX_BASE_ZERO 和 MCSPARSE_INDEX_BASE_ONE。
rowBlockDim	稀疏矩阵 C 的行数。
colBlockDim	稀疏矩阵 C 的列数。
pBuffer	由用户分配的缓冲区，在 csr2gebsr_bufferSize() 中返回其大小。

输出

bsrValC	<type> 矩阵 C 的 nnzb*rowBlockDim*colBlockDim 个非零元素的数组。
bsrRowPtrC	一个包含 mb+1 个元素的整数数组，其中包含矩阵 C 的每个块行的起始位置，以及最后一个块行的结束位置加一。
bsrColIndC	整数数组包含矩阵 C 的 nnzb 个非零块的列索引。
nnzTotalDevHostPtr	矩阵 C 的非零块总数。指针 nnzTotalDevHostPtr 可以指向设备内存或主机内存。

有关返回状态的描述，请参见mcsparseStatus_t。

13.2 mcsparse<t>coo2csr()

```
mcsparseStatus_t
mcsparseXcoo2csr(mcsparseHandle_t   handle,
                 const int*          cooRowInd,
                 int                 nnz,
                 int                 m,
                 int*                csrRowPtr,
                 mcsparseIndexBase_t idxBase)
```

此函数将包含未压缩的行索引数组 (对应 COO 格式) 转换为包含压缩的行指针数组 (对应 CSR 格式)。

它还可以用于将包含未压缩列索引数组 (对应 COO 格式) 转换为包含列指针的数组 (对应 CSC 格式)。

- 该例程不需要额外的存储空间。
- 该例程支持异步执行。

输入

handle	处理 mcSPARSE 库上下文的句柄。
cooRowInd	整型数组，包含 nnz 个未压缩的行索引。
nnz	稀疏矩阵的非零个数 (也就是数组 cooRowInd 的长度)
m	矩阵 A 的行数。
idxBase	MCSPARSE_INDEX_BASE_ZERO 或 MCSPARSE_INDEX_BASE_ONE。

输出

csrRowPtr	由 m+1 个元素组成的整型数组，包含每行的起始位置和最后一行的结束位置加一。

有关返回状态的描述，请参见mcsparseStatus_t。

13.3 mcsparse<t>csc2dense()

```
mcsparseStatus_t
mcsparseScsc2dense(mcsparseHandle_t         handle,
                   int                      m,
                   int                      n,
                   const mcsparseMatDescr_t descrA,
                   const float*             cscValA,
                   const int*               cscRowIndA,
                   const int*               cscColPtrA,
                   float*                   A,
                   int                      lda)

mcsparseStatus_t
mcsparseDcsc2dense(mcsparseHandle_t         handle,
                   int                      m,
                   int                      n,
                   const mcsparseMatDescr_t descrA,
                   const double*            cscValA,
                   const int*               cscRowIndA,
                   const int*               cscColPtrA,
                   double*                  A,
                   int                      lda)

mcsparseStatus_t
mcsparseCcsc2dense(mcsparseHandle_t         handle,
                   int                      m,
                   int                      n,
                   const mcsparseMatDescr_t descrA,
                   const mcComplex*         cscValA,
                   const int*               cscRowIndA,
                   const int*               cscColPtrA,
                   mcComplex*               A,
                   int                      lda)

mcsparseStatus_t
mcsparseZcsc2dense(mcsparseHandle_t         handle,
                   int                      m,
                   int                      n,
                   const mcsparseMatDescr_t descrA,
                   const mcDoubleComplex*   cscValA,
                   const int*               cscRowIndA,
                   const int*               cscColPtrA,
                   mcDoubleComplex*         A,
                   int                      lda)
```

此函数将 CSC 格式的稀疏矩阵转换为稠密格式的矩阵 A，其中 CSC 格式通过 cscValA，cscColPtrA 和 cscRowIndA 三个数组定义。稠密矩阵 A 用稀疏矩阵的值填充，其余地方用零填充。

- 该例程不需要额外的存储空间。
- 该例程支持异步执行。

输入

handle	处理 mcSPARSE 库上下文的句柄。
m	矩阵 A 的行数。
n	矩阵 A 的列数。
descrA	矩阵 A 的描述符。支持的矩阵类型为 MCSPARSE_MATRIX_TYPE_GENERAL。同时，支持的索引基数是 MCSPARSE_INDEX_BASE_ZERO 和 MCSPARSE_INDEX_BASE_ONE。
cscValA	<type> 数组，包含矩阵 A 中 nnz 个非零元素。
cscRowIndA	整型数组，包含矩阵 A 中 nnz 个非零元素的行索引。
cscColPtrA	由 n+1 个元素组成的整型数组，包含每行的起始位置和最后一列的结束位置加一。
lda	稠密数组 A 的主维度。

输出

A	维度为 (lda, n) 的数组，其中填充了稀疏矩阵的值。

有关返回状态的描述，请参见mcsparseStatus_t。

13.4 mcsparse<t>csr2bsr()

```
mcsparseStatus_t
mcsparseXcsr2bsrNnz(mcsparseHandle_t        handle,
                    mcsparseDirection_t     dir,
                    int                     m,
                    int                     n,
                    const mcsparseMatDescr_t descrA,
                    const int*              csrRowPtrA,
                    const int*              csrColIndA,
                    int                     blockDim,
                    const mcsparseMatDescr_t descrC,
                    int*                    bsrRowPtrC,
                    int*                    nnzTotalDevHostPtr)

mcsparseStatus_t
mcsparseScsr2bsr(mcsparseHandle_t         handle,
                 mcsparseDirection_t      dir,
                 int                      m,
                 int                      n,
                 const mcsparseMatDescr_t descrA,
                 const float*             csrValA,
                 const int*               csrRowPtrA,
                 const int*               csrColIndA,
                 int                      blockDim,
                 const mcsparseMatDescr_t descrC,
                 float*                   bsrValC,
                 int*                     bsrRowPtrC,
                 int*                     bsrColIndC)

mcsparseStatus_t
```

(下页继续)

(续上页)

```
mcsparseDcsr2bsr(mcsparseHandle_t         handle,
                 mcsparseDirection_t      dir,
                 int                      m,
                 int                      n,
                 const mcsparseMatDescr_t descrA,
                 const double*            csrValA,
                 const int*               csrRowPtrA,
                 const int*               csrColIndA,
                 int                      blockDim,
                 const mcsparseMatDescr_t descrC,
                 double*                  bsrValC,
                 int*                     bsrRowPtrC,
                 int*                     bsrColIndC)

mcsparseStatus_t
mcsparseCcsr2bsr(mcsparseHandle_t         handle,
                 mcsparseDirection_t      dir,
                 int                      m,
                 int                      n,
                 const mcsparseMatDescr_t descrA,
                 const mcComplex*         csrValA,
                 const int*               csrRowPtrA,
                 const int*               csrColIndA,
                 int                      blockDim,
                 const mcsparseMatDescr_t descrC,
                 mcComplex*               bsrValC,
                 int*                     bsrRowPtrC,
                 int*                     bsrColIndC)

mcsparseStatus_t
mcsparseZcsr2bsr(mcsparseHandle_t         handle,
                 mcsparseDirection_t      dir,
                 int                      m,
                 int                      n,
                 const mcsparseMatDescr_t descrA,
                 const mcDoubleComplex*   csrValA,
                 const int*               csrRowPtrA,
                 const int*               csrColIndA,
                 int                      blockDim,
                 const mcsparseMatDescr_t descrC,
                 mcDoubleComplex*         bsrValC,
                 int*                     bsrRowPtrC,
                 int*                     bsrColIndC)
```

此函数将 CSR 格式的稀疏矩阵转换为 BSR 格式的稀疏矩阵，其中 CSR 格式通过三个数组 `csrValA`，`csrRowPtrA` 和 `csrColIndA` 定义，BSR 格式通过数组 `bsrValC`，`bsrRowPtrC`，和 `bsrColIndC` 定义。`A` 是一个 `m*n` 稀疏矩阵。`A` 的 BSR 格式具有 `mb` 块行，`nb` 块列和 `nnzb` 非零块，其中 `mb=((m+blockDim-1)/blockDim)` 和 `nb=(n+blockDim-1)/blockDim`。

如果 `m` 或 `n` 不是 `blockDim` 的倍数，则填充零。

mcSPARSE 中的转换通过以下操作实现。

首先，用户分配 `mb+1` 元素的 `bsrRowPtrC`，并使用函数 `mcsparseXcsr2bsrNnz()` 来确定每个块行的非零块列的数量。其次，用户从 `(nnzb=*nnzTotalDevHostPtr)` 或 `(nnzb=bsrRowPtrC[mb]-bsrRowPtrC[0])` 中收集 `nnzb` (矩阵 `C` 的非零块列数) 并分配 `nnzb*blockDim*blockDim` 个 `bsrValC` 元素和 `nnzb` 个 `bsrColIndC` 元素。最后调用函数 `mcsparse[S|D|C|Z]csr2bsr90` 来完成转换。

一般程序如下：

```
// 给定 CSR 格式 (csrRowPtrA, csrcolIndA, csrValA) 和
// BSR 格式的块按列主序的顺序存储。
mcsparseDirection_t dir = MCSPARSE_DIRECTION_COLUMN;
int base, nnzb;
int mb = (m + blockDim-1)/blockDim;
mcMalloc((void**)&bsrRowPtrC, sizeof(int) *(mb+1));
// nnzTotalDevHostPtr 指向主机内存
int *nnzTotalDevHostPtr = &nnzb;
mcsparseXcsr2bsrNnz(handle, dir, m, n,
          descrA, csrRowPtrA, csrColIndA,
          blockDim,
          descrC, bsrRowPtrC,
          nnzTotalDevHostPtr);
if (NULL != nnzTotalDevHostPtr){
      nnzb = *nnzTotalDevHostPtr;
}else{
      mcMemcpy(&nnzb, bsrRowPtrC+mb, sizeof(int),
↪mcMemcpyDeviceToHost);
      mcMemcpy(&base, bsrRowPtrC, sizeof(int),
↪mcMemcpyDeviceToHost);
      nnzb -= base;
}
mcMalloc((void**)&bsrColIndC, sizeof(int)*nnzb);
mcMalloc((void**)&bsrValC, sizeof(float)*(blockDim*blockDim)*nnzb);
mcsparseScsr2bsr(handle, dir, m, n,
          descrA,
          csrValA, csrRowPtrA, csrColIndA,
          blockDim,
          descrC,
          bsrValC, bsrRowPtrC, bsrColIndC);
```

例程 `mcsparse<t>csr2bsr()` 具有以下特性：

- 如果 `blockDim > 16`，此函数需要在内部分配临时的额外存储空间。
- 如果 `blockDim != 1` 并且流式有序内存分配器可用，则例程支持异步执行

例程 `mcsparseXcsr2bsrNnz()` 具有以下特性：

- 此函数需要内部分配的临时额外存储空间。
- 如果流式有序内存分配器可用，则该例程支持异步执行。

输入

handle	处理 mcSPARSE 库上下文的句柄。
dir	块的存储格式 MCSPARSE_DIRECTION_ROW 或 MCSPARSE_DIRECTION_COLUMN。
m	稀疏矩阵 A 的行数。
n	稀疏矩阵 A 的列数。
descrA	矩阵 A 的描述符。
csrValA	<type> 数组，包含 nnz(=csrRowPtrA[m]-csrRowPtr[0]) 个矩阵 A 的非零元素。
csrRowPtrA	由 m+1 个元素组成的整型数组，包含每行的起始位置和最后一行的结束位置加一。
csrColIndA csrColIndA	整型数组，包含矩阵 A 中 nnz 个非零元素的列索引。
blockDim	稀疏矩阵 A 的块维度。blockDim 的取值范围在 1 到 min(m,n) 之间。
descrC	矩阵 C 的描述符。

输出

bsrValC	<type> 数组，包含 nnzb*blockDim*blockDim 个矩阵 C 的非零元素。
bsrRowPtrC	由 mb+1 个元素组成的整型数组，包含每行的起始位置和最后一行的结束位置加一。
bsrColIndC	整型数组，包含矩阵 C 中 nnzb 个非零块的列索引。
nnzTotalDevHostPtr	设备或主机内存中非零元素的总数。它等于 (bsrRowPtrC[mb]-bsrRowPtrC[0])。

有关返回状态的描述，请参见mcsparseStatus_t。

13.5 mcsparse<t>csr2coo()

```
mcsparseStatus_t
mcsparseXcsr2coo(mcsparseHandle_t     handle,
                 const int*           csrRowPtr,
                 int                  nnz,
                 int                  m,
                 int*                 cooRowInd,
                 mcsparseIndexBase_t  idxBase)
```

此函数将包含压缩的行指针数组 (对应 CSR 格式) 转换为未压缩的行索引数组 (对应 COO 格式)。

此函数还可以用于将包含压缩的列索引数组 (对应 CSC 格式) 转换为包含未压缩列索引的数组 (对应 COO 格式)。

- 该例程不需要额外的存储空间。
- 该例程支持异步执行。

输入

handle	处理 mcSPARSE 库上下文的句柄。
csrRowPtr	由 m+1 个元素组成的整型数组，包含每行的起始位置和最后一行的结束位置加一。
nnz	稀疏矩阵的非零个数 (也是数组 cooRowInd 的长度)。
m	矩阵 A 的列数。
idxBase	MCSPARSE_INDEX_BASE_ZERO 或 MCSPARSE_INDEX_BASE_ONE。

输出

cooRowInd	整型数组，包含 nnz 个未压缩的行索引。

有关返回状态的描述，请参见mcsparseStatus_t。

13.6 mcsparse<t>csr2dense()

```
mcsparseStatus_t
mcsparseScsr2dense(mcsparseHandle_t          handle,
                   int                       m,
                   int                       n,
                   const mcsparseMatDescr_t  descrA,
```

(下页继续)

(续上页)

```
                      const float*             csrValA,
                      const int*               csrRowPtrA,
                      const int*               csrColIndA,
                      float*                   A,
                      int                      lda)

mcsparseStatus_t
mcsparseDcsr2dense(mcsparseHandle_t         handle,
                      int                      m,
                      int                      n,
                      const mcsparseMatDescr_t descrA,
                      const double*            csrValA,
                      const int*               csrRowPtrA,
                      const int*               csrColIndA,
                      double*                  A,
                      int                      lda)

mcsparseStatus_t
mcsparseCcsr2dense(mcsparseHandle_t         handle,
                      int                      m,
                      int                      n,
                      const mcsparseMatDescr_t descrA,
                      const mcComplex*         csrValA,
                      const int*               csrRowPtrA,
                      const int*               csrColIndA,
                      mcComplex*               A,
                      int                      lda)

mcsparseStatus_t
mcsparseZcsr2dense(mcsparseHandle_t         handle,
                      int                      m,
                      int                      n,
                      const mcsparseMatDescr_t descrA,
                      const mcDoubleComplex*   csrValA,
                      const int*               csrRowPtrA,
                      const int*               csrColIndA,
                      mcDoubleComplex*         A,
                      int                      lda)
```

此函数将 CSR 格式的稀疏矩阵转换为稠密格式的矩阵 `A`，其中 CSR 格式通过三个数组 `csrValA`，`csrRowPtrA` 和 `csrColIndA` 定义。稠密矩阵 `A` 用稀疏矩阵的值填充，其余地方用零填充。

- 该例程不需要额外的存储空间。
- 该例程支持异步执行。

输入

handle	处理 mcSPARSE 库上下文的句柄。
m	矩阵的行数。
n	矩阵的列数。
descrA	矩阵的描述符。支持的矩阵大小为 MCSPARSE_MATRIX_TYPE_GENERAL 同时也支持索引基数为 MCSPARSE_INDEX_BASE_ZERO 和 MCSPARSE_INDEX_BASE_ONE。
csrValA	<type> 数组，包含矩阵中 nnz 个的非零元素。
csrRowPtrA	由 m+1 个元素组成的整型数组，包含每行起始位置和最后一行的结束位置加一。
csrColIndA	整型数组，包含矩阵中 nnz 个非零元素的列索引。
lda	数组矩阵 A 主维度

输出

A	维度为 (lda,n) 的数组。用稀疏矩阵的值填充。

有关返回状态的描述，请参见mcsparseStatus_t。

13.7 mcsparse<t>csr2csr_compress()

```
mcsparseStatus_t
mcsparseScsr2csr_compress(mcsparseHandle_t         handle,
                          int                      m,
                          int                      n,
                          const mcsparseMatDescr_t descrA,
                          const float*             csrValA,
                          const int*               csrColIndA,
                          const int*               csrRowPtrA,
                          int                      nnzA,
                          const int*               nnzPerRow,
                          float*                   csrValC,
                          int*                     csrColIndC,
                          int*                     csrRowPtrC,
                          float                    tol)

mcsparseStatus_t
mcsparseDcsr2csr_compress(mcsparseHandle_t         handle,
                          int                      m,
                          int                      n,
                          const mcsparseMatDescr_t descrA,
                          const double*            csrValA,
                          const int*               csrColIndA,
                          const int*               csrRowPtrA,
                          int                      nnzA,
                          const int*               nnzPerRow,
                          double*                  csrValC,
                          int*                     csrColIndC,
                          int*                     csrRowPtrC,
                          double                   tol)

mcsparseStatus_t
mcsparseCcsr2csr_compress(mcsparseHandle_t         handle,
                          int                      m,
```

(下页继续)

(续上页)

```
                            int                         n,
                            const mcsparseMatDescr_t    descrA,
                            const mcComplex*            csrValA,
                            const int*                  csrColIndA,
                            const int*                  csrRowPtrA,
                            int                         nnzA,
                            const int*                  nnzPerRow,
                            mcComplex*                  csrValC,
                            int*                        csrColIndC,
                            int*                        csrRowPtrC,
                            mcComplex                   tol)

mcsparseStatus_t
mcsparseZcsr2csr_compress(mcsparseHandle_t       handle,
                            int                         m,
                            int                         n,
                            const mcsparseMatDescr_t    descrA,
                            const mcDoubleComplex*      csrValA,
                            const int*                  csrColIndA,
                            const int*                  csrRowPtrA,
                            int                         nnzA,
                            const int*                  nnzPerRow,
                            mcDoubleComplex*            csrValC,
                            int*                        csrColIndC,
                            int*                        csrRowPtrC,
                            mcDoubleComplex             tol)
```

此函数将 CSR 格式的稀疏矩阵压缩成压缩的 CSR 格式。给定一个稀疏矩阵 A 和一个非负阈值，此函数返回一个稀疏矩阵 C。

该实现通过以下操作进行转换：

首先，用户分配 m+1 个元素的 csrRowPtrC，并使用函数 `mcsparse<t>nnz_compress()` 确定 nnzPerRow（每行非零列的数量）和 nnzC（非零总数）。其次，用户分配 nnzC 个元素的 csrValC 和 nnzC 个整型的 csrColIndC。最后调用函数 `mcsparse<t>csr2csr_compress()` 来完成转换。

- 此函数需要内部分配的临时额外存储空间。
- 如果流式有序内存分配器可用，则该例程支持异步执行。

输入

handle	处理 mcSPARSE 库上下文的句柄。
m	矩阵的行数。
n	矩阵的列数。
descrA	矩阵的描述符。支持的矩阵大小为 MCSPARSE_MATRIX_TYPE_GENERAL 同时也支持索引基数为 MCSPARSE_INDEX_BASE_ZERO 和 MCSPARSE_INDEX_BASE_ONE。
csrValA	<type> 数组，包含矩阵中 nnz 个元素。
csrColIndA	整型数组，包含矩阵中 nnz 个元素的列索引。
csrRowPtrA	由 m+1 个元素组成的整型数组，其中包含每行的起始位置和最后一行的结束位置加一。
nnzA	矩阵的非零元素数。
nnzPerRow	该数组包含按行存储压缩矩阵中的元素数量。
tol	在输入时，它包含用于压缩的非负容差值。在压缩过程中。矩阵 A 中小于或等于该值的任何值都将被丢弃。

输出

csrValC	在输出时，该数组包含存储在压缩矩阵中的元素的类型值 Size =nnzC。
csrColIndC	在输出时，该整型数组包含存储在压缩矩阵中元素的列索引。Size = nnzC。
csrRowPtrC	在输出时，该整型数组包含存储在压缩矩阵中的元素的行指针。Size = m+1。

有关返回状态的描述，请参见mcsparseStatus_t。

下面代码段演示如何使用此 API。

```
#include <stdio.h>
#include <sys/time.h>
#include <mcsparse.h>

#define ERR_NE(X,Y) do { if ((X) != (Y)) { \
fprintf(stderr,"Error in %s at %s:%d\n",__func__,__FILE__,__LINE__
↪);\
exit(-1);}} while(0)
#define MACA_CALL(X) ERR_NE((X),macaSuccess)
#define MCSPARSE_CALL(X) ERR_NE((X),MCSPARSE_STATUS_SUCCESS)
int main(){
      int m = 6, n = 5;
      mcsparseHandle_t  handle;
      MCSPARSE_CALL( mcsparseCreate(&handle) );
      mcsparseMatDescr_t descrX;
      MCSPARSE_CALL(mcsparseCreateMatDescr(&descrX));
      // 初始化稀疏矩阵
      float *X;
      MACA_CALL(mcMallocManaged( &X, sizeof(float) * m * n ));
      memset( X, 0, sizeof(float) * m * n );
      X[0 + 0*m] = 1.0;  X[0 + 1*m] = 3.0;
      X[1 + 1*m] = -4.0; X[1 + 2*m] = 5.0;
      X[2 + 0*m] = 2.0;  X[2 + 3*m] = 7.0;  X[2 + 4*m] = 8.0;
      X[3 + 2*m] = 6.0;  X[3 + 4*m] = 9.0;
      X[4 + 3*m] = 3.5;  X[4 + 4*m] = 5.5;
      X[5 + 0*m] = 6.5;  X[5 + 2*m] = -9.9;
      // 为了 mcsparseSdense2csr()，初始化 total_nnz 和 nzPerRowX
      int total_nnz = 13;
      int *nnzPerRowX;
      MACA_CALL( mcMallocManaged( &nnzPerRowX, sizeof(int) * m ));
      nnzPerRowX[0] = 2;  nnzPerRowX[1] = 2;  nnzPerRowX[2] = 3;
      nnzPerRowX[3] = 2;  nnzPerRowX[4] = 2;  nnzPerRowX[5] = 2;

      float *csrValX;
      int *csrRowPtrX;
      int *csrColIndX;
      MACA_CALL( mcMallocManaged( &csrValX, sizeof(float) * total_
↪nnz) );
      MACA_CALL( mcMallocManaged( &csrRowPtrX, sizeof(int) *
↪(m+1))) ;
      MACA_CALL( mcMallocManaged( &csrColIndX, sizeof(int) * total_
↪nnz)) ;
```

在调用此 API 之前，请调用两个 API 来准备输入。

```
/** 调用 mcsparseSdense2csr 来生成 CSR 格式，作为
      mcsparseScsr2csr_compress 的输入  **/
      MCSPARSE_CALL( mcsparseSdense2csr( handle, m, n, descrX, X,
                                         m, nnzPerRowX, csrValX,
```

(下页继续)

(续上页)

```
                                        csrRowPtrX, csrColIndX )) ;
    float tol = 3.5;
    int *nnzPerRowY;
    int *testNNZTotal;
    MACA_CALL (mcMallocManaged( &nnzPerRowY, sizeof(int) * m ));
    MACA_CALL (mcMallocManaged( &testNNZTotal, sizeof(int)));
    memset( nnzPerRowY, 0, sizeof(int) * m );
    // mcsparseSnnz_compress 生成 nnzPerRowY 和 testNNZTotal
    MCSPARSE_CALL( mcsparseSnnz_compress(handle, m, descrX,
↪csrValX,
                                         csrRowPtrX, nnzPerRowY,
                                         testNNZTotal, tol));

    float *csrValY;
    int *csrRowPtrY;
    int *csrColIndY;
    MACA_CALL( mcMallocManaged( &csrValY, sizeof(float) *
↪(*testNNZTotal)));
    MACA_CALL( mcMallocManaged( &csrRowPtrY, sizeof(int) *
↪(m+1)));
    MACA_CALL( mcMallocManaged( &csrColIndY, sizeof(int) *
↪(*testNNZTotal)));

    MCSPARSE_CALL( mcsparseScsr2csr_compress( handle, m, n,
↪descrX, csrValX,
                                              csrColIndX,
↪csrRowPtrX,
                                              total_nnz,
↪nnzPerRowY,
                                              csrValY,
↪csrColIndY,
                                              csrRowPtrY, tol));
    /* 期望结果
    nnzPerRowY:  0 2 2 2 1 2
    csrValY:     -4 5 7 8 6 9 5.5 6.5 -9.9
    csrColIndY:  1 2 3 4 2 4 4 0 2
    csrRowPtrY:  0 0 2 4 6 7 9
    */
    mcFree(X);
    mcsparseDestroy(handle);
    mcFree(nnzPerRowX);
    mcFree(csrValX);
    mcFree(csrRowPtrX);
    mcFree(csrColIndX);
    mcFree(csrValY);
    mcFree(nnzPerRowY);
    mcFree(testNNZTotal);
    mcFree(csrRowPtrY);
    mcFree(csrColIndY);
    return 0;
}
```

13.8 mcsparse<t>dense2csc()

```
mcsparseStatus_t
mcsparseSdense2csc(mcsparseHandle_t         handle,
                   int                      m,
                   int                      n,
                   const mcsparseMatDescr_t descrA,
                   const float*             A,
                   int                      lda,
                   const int*               nnzPerCol,
                   float*                   cscValA,
                   int*                     cscRowIndA,
                   int*                     cscColPtrA)

mcsparseStatus_t
mcsparseDdense2csc(mcsparseHandle_t         handle,
                   int                      m,
                   int                      n,
                   const mcsparseMatDescr_t descrA,
                   const double*            A,
                   int                      lda,
                   const int*               nnzPerCol,
                   double*                  cscValA,
                   int*                     cscRowIndA,
                   int*                     cscColPtrA)

mcsparseStatus_t
mcsparseCdense2csc(mcsparseHandle_t         handle,
                   int                      m,
                   int                      n,
                   const mcsparseMatDescr_t descrA,
                   const mcComplex*         A,
                   int                      lda,
                   const int*               nnzPerCol,
                   mcComplex*               cscValA,
                   int*                     cscRowIndA,
                   int*                     cscColPtrA)

mcsparseStatus_t
mcsparseZdense2csc(mcsparseHandle_t         handle,
                   int                      m,
                   int                      n,
                   const mcsparseMatDescr_t descrA,
                   const mcDoubleComplex*   A,
                   int                      lda,
                   const int*               nnzPerCol,
                   mcDoubleComplex*         cscValA,
                   int*                     cscRowIndA,
                   int*                     cscColPtrA)
```

此函数将稠密格式的矩阵 `A` 转换为 CSC 格式的稀疏矩阵。所有参数都假定是由用户预先分配，并且数组是基于 `nnzPerCol` 来填充的，可以使用 `mcsparse<t>nnz()` 来进行预处理计算。

- 此函数需要内部分配的临时额外存储空间。
- 如果流式有序内存分配器可用，则该例程支持异步执行。

输入

handle	处理 mcSPARSE 库上下文的句柄。
m	矩阵 A 的行数。
n	矩阵 A 的列数。
descrA	矩阵 A 的描述符。支持的矩阵大小为 MCSPARSE_MATRIX_TYPE_GENERAL。同时也支持索引基数为 MCSPARSE_INDEX_BASE_ZERO 和 MCSPARSE_INDEX_BASE_ONE。
A	维度为 (lda, n) 的数组。
lda	稠密数组 A 的主维度。
nnzPerCol	大小为 n 的数组，包含每列非零元素的数量。

输出

cscValA	<type> 数组，包含矩阵 A 中 nnz 个的非零元素。只有当 copyValues 设置为 MCSPARSE_ACTION_NUMERIC 时，它才会被填写。
cscRowIndA	整型数组，包含矩阵 A 中 nnz 个非零元素的行索引。
cscColPtrA	由 n+1 个元素组成的整型数组，包含每一列的起始位置和最后一列的结束位置加一。

有关返回状态的描述，请参见mcsparseStatus_t。

13.9 mcsparse<t>dense2csr()

```
mcsparseStatus_t
mcsparseSdense2csr(mcsparseHandle_t          handle,
                   int                       m,
                   int                       n,
                   const mcsparseMatDescr_t  descrA,
                   const float*              A,
                   int                       lda,
                   const int*                nnzPerRow,
                   float*                    csrValA,
                   int*                      csrRowPtrA,
                   int*                      csrColIndA)

mcsparseStatus_t
mcsparseDdense2csr(mcsparseHandle_t          handle,
                   int                       m,
                   int                       n,
                   const mcsparseMatDescr_t  descrA,
                   const double*             A,
                   int                       lda,
                   const int*                nnzPerRow,
                   double*                   csrValA,
                   int*                      csrRowPtrA,
                   int*                      csrColIndA)

mcsparseStatus_t
mcsparseCdense2csr(mcsparseHandle_t          handle,
                   int                       m,
                   int                       n,
                   const mcsparseMatDescr_t  descrA,
                   const mcComplex*          A,
                   int                       lda,
```

(下页继续)

(续上页)

```
                      const int*                nnzPerRow,
                      mcComplex*                csrValA,
                      int*                      csrRowPtrA,
                      int*                      csrColIndA)

mcsparseStatus_t
mcsparseZdense2csr(mcsparseHandle_t          handle,
                      int                       m,
                      int                       n,
                      const mcsparseMatDescr_t  descrA,
                      const mcDoubleComplex*    A,
                      int                       lda,
                      const int*                nnzPerRow,
                      mcDoubleComplex*          csrValA,
                      int*                      csrRowPtrA,
                      int*                      csrColIndA)
```

此函数将稠密格式的矩阵 A 转换为 CSR 格式的稀疏矩阵。假设用户预先分配了所有参数，并且数组是根据 nnzPerRow 填充的，这个值可以通过 mcsparse<t>nnz() 进行预先计算。

此函数不需要额外的存储空间。它在与主机异步执行，可能在结果准备好之前将控制权返回给主机上的应用程序。

- 此函数需要内部分配的临时额外存储空间
- 如果流式有序内存分配器可用，则该例程支持异步执行

输入

handle	处理 mcSPARSE 库上下文的句柄。
m	矩阵 A 的行数。
n	矩阵 A 的列数。
descrA	矩阵 A 的描述符。支持的矩阵类型是 MCSPARSE_MATRIX_TYPE_GENERAL。同时，支持的基准索引是 MCSPARSE_INDEX_BASE_ZERO 和 MCSPARSE_INDEX_BASE_ONE。
A	大小为 (lda, n) 的数组。
lda	稠密数组 A 的主维度。
nnzPerRow	大小为 n 的数组，包含每行非零元素的数量。

输出

csrValA	<type> 数组，包含矩阵 A 中 nnz 个非零元素。
csrRowPtrA	整型数组，包含 m+1 个元素，其中每个元素表示每一行的起始位置和最后一行的结束位置加一。
csrColIndA	整数数组，包含矩阵 A 中 nnz 个非零元素的列索引。

有关返回状态的描述，请参见mcsparseStatus_t。

13.10 mcsparse<t>nnz()

```
mcsparseStatus_t
mcsparseSnnz(mcsparseHandle_t        handle,
             mcsparseDirection_t     dirA,
             int                     m,
```

(下页继续)

(续上页)

```
                int                         n,
                const mcsparseMatDescr_t    descrA,
                const float*                A,
                int                         lda,
                int*                        nnzPerRowColumn,
                int*                        nnzTotalDevHostPtr)

mcsparseStatus_t
mcsparseDnnz(mcsparseHandle_t         handle,
                mcsparseDirection_t     dirA,
                int                       m,
                int                       n,
                const mcsparseMatDescr_t  descrA,
                const double*             A,
                int                       lda,
                int*                      nnzPerRowColumn,
                int*                      nnzTotalDevHostPtr)

mcsparseStatus_t
mcsparseCnnz(mcsparseHandle_t         handle,
                mcsparseDirection_t     dirA,
                int                       m,
                int                       n,
                const mcsparseMatDescr_t  descrA,
                const mcComplex*          A,
                int                       lda,
                int*                      nnzPerRowColumn,
                int*                      nnzTotalDevHostPtr)

mcsparseStatus_t
mcsparseZnnz(mcsparseHandle_t         handle,
                mcsparseDirection_t     dirA,
                int                       m,
                int                       n,
                const mcsparseMatDescr_t  descrA,
                const mcDoubleComplex*    A,
                int                       lda,
                int*                      nnzPerRowColumn,
                int*                      nnzTotalDevHostPtr)
```

此函数计算稠密矩阵中每行或每列的非零元素数量，以及稠密矩阵中总的非零元素数量。

- 此函数需要内部分配的临时额外存储空间
- 如果流式有序内存分配器可用，则该例程支持异步执行

输入

handle	处理 mcSPARSE 库上下文的句柄。
dirA	方向参数，指定是按 MCSPARSE_DIRECTION_ROW 还是按 MCSPARSE_DIRECTION_COLUMN 计算非零元素数量。
m	矩阵 A 的行数。
n	矩阵 A 的列数。
descrA	矩阵 A 的描述符。支持的矩阵类型是 MCSPARSE_MATRIX_TYPE_GENERAL。同时，支持的基准索引是 MCSPARSE_INDEX_BASE_ZERO 和 MCSPARSE_INDEX_BASE_ONE。
A	大小为 (lda, n) 的数组。
lda	稠密数组 A 的主维度。

输出

nnzPerRowColumn	大小为 m 或 n 的数组，分别包含每行或每列的非零元素数量。
nnzTotalDevHostPtr	非零元素的总数，存储在设备或主机内存中。

有关返回状态的描述，请参见mcsparseStatus_t。

13.11 mcsparseCreateIdentityPermutation()

```
mcsparseStatus_t
mcsparseCreateIdentityPermutation(mcsparseHandle_t handle,
                                  int              n,
                                  int*             p);
```

此函数创建一个单位映射。输出参数 p 通过 p = 0:1:(n-1) 表示这样的映射。

此函数通常与 coosort、csrsort 和 cscsort 一起使用。

- 该例程不需要额外的存储空间
- 该例程支持异步执行

输入

参数	设备或主机	含义
handle	主机	处理 mcSPARSE 库上下文的句柄。
n	主机	映射的大小。

输出

参数	设备或主机	含义
p	设备	整数数组，大小为 n。

有关返回状态的描述，请参见mcsparseStatus_t。

13.12 mcsparseXcoosort()

```
mcsparseStatus_t
mcsparseXcoosort_bufferSizeExt(mcsparseHandle_t handle,
                               int              m,
                               int              n,
                               int              nnz,
                               const int*       cooRows,
                               const int*       cooCols,
                               size_t*          pBufferSizeInBytes)
```

```
mcsparseStatus_t
mcsparseXcoosortByRow(mcsparseHandle_t handle,
                      int              m,
                      int              n,
                      int              nnz,
                      int*             cooRows,
                      int*             cooCols,
```

(下页继续)

(续上页)

```
                          int*              P,
                          void*             pBuffer)

mcsparseStatus_t
mcsparseXcoosortByColumn(mcsparseHandle_t handle,
                             int               m,
                             int               n,
                             int               nnz,
                             int*              cooRows,
                             int*              cooCols,
                             int*              P,
                             void*             pBuffer);
```

此函数对 COO 格式进行排序。排序是原地进行的。用户可以按行或按列对矩阵进行排序。A 是一个大小为 m×n 的稀疏矩阵，由三个数组 cooVals、cooRows 和 cooCols 以 COO 存储格式定义。

对于矩阵的基准索引没有假设。coosort 在有符号整数上使用稳定排序，因此 cooRows 或 cooCols 的值可以是负数。

此函数 coosort() 需要通过 coosort_bufferSizeExt() 返回缓冲区大小。pBuffer 的地址必须是 128 字节的倍数。如果不是，则返回 MCSPARSE_STATUS_INVALID_VALUE。

参数 P 同时用作输入和输出。如果用户想要计算排序后的 cooVal，则必须在 coosort() 之前将 P 设置为 0:1:(nnz-1)，然后在 coosort() 之后，新的排序值数组满足 cooVal_sorted = cooVal(P)。

注解： 维度 m 和 n 没有使用。如果用户不知道 m 或 n 的值，只需传递一个正数值。这种情况通常发生在用户首先只读取一个 COO 数组，随后需要决定维数 m 或 n 时。

- 如果 pBuffer != NULL，该例程不需要额外的存储空间
- 如果流式有序内存分配器可用，则该例程支持异步执行

输入

参数	设备或主机	含义
handle	主机	处理 mcSPARSE 库上下文的句柄。
m	主机	矩阵 A 的行数。
n	主机	矩阵 A 的列数。
nnz	主机	矩阵 A 中非零元素的数量。
cooRows	设备	整数数组，包含 nnz 个未排序的 A 的行索引。
cooCols	设备	整数数组，包含 nnz 个未排序的 A 的列索引。
P	设备	整数数组，包含 nnz 个未排序元素的映射索引。为构建 csrVal，用户需要设置 P=0:1:(nnz-1)。
pBuffer	设备	用户分配的缓冲区；缓冲区的大小由 coosort_bufferSizeExt() 返回。

输出

参数	设备或主机	含义
cooRows	设备	整数数组，包含 nnz 个排序后的 A 的行索引。
cooCols	设备	整数数组，包含 nnz 个排序后的 A 的列索引。
P	设备	整数数组，包含 nnz 个排序后的映射索引。
pBufferSizeInBytes	主机	缓冲区的字节数。

有关返回状态的描述，请参见mcsparseStatus_t。

13.13 mcsparseXcsrsort()

```
mcsparseStatus_t
mcsparseXcsrsort_bufferSizeExt(mcsparseHandle_t handle,
                               int              m,
                               int              n,
                               int              nnz,
                               const int*       csrRowPtr,
                               const int*       csrColInd,
                               size_t*          pBufferSizeInBytes)
```

```
mcsparseStatus_t
mcsparseXcsrsort(mcsparseHandle_t         handle,
                 int                      m,
                 int                      n,
                 int                      nnz,
                 const mcsparseMatDescr_t descrA,
                 const int*               csrRowPtr,
                 int*                     csrColInd,
                 int*                     P,
                 void*                    pBuffer)
```

此函数用于对 CSR 格式的稀疏矩阵进行排序。排序是在原地进行的，并且是稳定的。

矩阵类型被隐式地视为 MCSPARSE_MATRIX_TYPE_GENERAL。即，任何对称性质都会被忽略。

此函数 csrsort() 需要通过 csrsort_bufferSizeExt() 返回缓冲区大小。缓冲区的地址必须是 128 字节的倍数。如果不是，将返回 MCSPARSE_STATUS_INVALID_VALUE。

参数 P 既是输入参数又是输出参数。如果用户想要计算排序后的 csrVal，则在调用 csrsort() 之前，必须将 P 设置为 0:1:(nnz-1)，并且在调用 csrsort() 完成后，新的排序值数组将满足 csrVal_sorted = csrVal(P)。

总体步骤如下：

```
// A 是 3x3 稀疏矩阵，基为 0
//     | 1 2 3 |
// A = | 4 5 6 |
//     | 7 8 9 |
const int m = 3;
const int n = 3;
const int nnz = 9;
csrRowPtr[m+1] = { 0, 3, 6, 9}; // 在设备上
csrColInd[nnz] = { 2, 1, 0, 0, 2,1, 1, 2, 0}; // 在设备上
csrVal[nnz] = { 3, 2, 1, 4, 6, 5, 8, 9, 7}; // 在设备上
size_t pBufferSizeInBytes = 0;
void *pBuffer = NULL;
int *P = NULL;

// 步骤 1：分配缓冲区
mcsparseXcsrsort_bufferSizeExt(handle, m, n, nnz, csrRowPtr,
↪csrColInd,
                               &pBufferSizeInBytes);
mcMalloc( &pBuffer, sizeof(char)* pBufferSizeInBytes);

// 步骤 2：设置置换向量 P 为单位映射
mcMalloc( (void**)&P, sizeof(int)*nnz);
mcsparseCreateIdentityPermutation(handle, nnz, P);
```

(下页继续)

(续上页)

```
// 步骤 3：对 CSR 格式进行排序
mcsparseXcsrsort(handle, m, n, nnz, descrA, csrRowPtr, csrColInd,
↪P, pBuffer);

// 步骤 4：收集排序后的 csrVal
mcsparseDgthr(handle, nnz, csrVal, csrVal_sorted, P, MCSPARSE_
↪INDEX_BASE_ZERO);
```

- 如果 pBuffer != NULL，该例程不需要额外的存储空间
- 如果流式有序内存分配器可用，则该例程支持异步执行

输入

参数	设备或主机	含义
handle	主机	处理 mcSPARSE 库上下文的句柄。
m	主机	矩阵 A 的行数。
n	主机	矩阵 A 的列数。
nnz	主机	矩阵 A 中非零元素的数量。
csrRowsPtr	设备	整数数组，包含 m+1 个元素，其中每个元素表示每一行的起始位置和最后一行的结束位置加一。
csrColInd	设备	整数数组，包含矩阵 A 中 nnz 个的未排序列索引。
P	设备	整数数组，包含 nnz 个未排序元素的映射索引。为构建 csrVal，用户需要设置 P=0:1:(nnz-1)。
pBuffer	设备	用户分配的缓冲区；缓冲区的大小由 coosort_bufferSizeExt() 返回。

输出

参数	设备或主机	含义
csrColInd	设备	整数数组，包含矩阵 A 中 nnz 个已排序的列索引。
P	设备	整数数组，包含 nnz 个已排序的映射索引。
pBufferSizeInBytes	主机	缓冲区的字节数。

有关返回状态的描述，请参见mcsparseStatus_t。

13.14 mcsparseXcscsort()

```
mcsparseStatus_t
mcsparseXcscsort_bufferSizeExt(mcsparseHandle_t handle,
                               int              m,
                               int              n,
                               int              nnz,
                               const int*       cscColPtr,
                               const int*       cscRowInd,
                               size_t*          pBufferSizeInBytes)
```

```
mcsparseStatus_t
mcsparseXcscsort(mcsparseHandle_t         handle,
                 int                       m,
                 int                       n,
                 int                       nnz,
```

(下页继续)

(续上页)

```
                const mcsparseMatDescr_t descrA,
                const int*               cscColPtr,
                int*                     cscRowInd,
                int*                     P,
                void*                    pBuffer)
```

此函数对 CSC 格式进行排序。排序是在原地进行的，并且是稳定的。

矩阵类型被隐式地视为 MCSPARSE_MATRIX_TYPE_GENERAL。即，任何对称性质都会被忽略。

此函数 csrsort() 需要通过 csrsort_bufferSizeExt() 返回缓冲区大小。缓冲区的地址必须是 128 字节的倍数。如果不是，将返回 MCSPARSE_STATUS_INVALID_VALUE。

参数 P 既是输入参数又是输出参数。如果用户想要计算排序后的 csrVal，则在调用 csrsort() 之前，必须将 P 设置为 0:1:(nnz-1)，并且在调用 csrsort() 完成后，新的排序值数组将满足 csrVal_sorted = csrVal(P)。

总体步骤如下：

```
// A 是 3x3 稀疏矩阵，基为 0
//     | 1 2  |
// A = | 4 0  |
//     | 0 8  |
const int m = 3;
const int n = 2;
const int nnz = 4;
cscColPtr[n+1] = { 0, 2, 4}; // 在设备上
cscRowInd[nnz] = { 1, 0, 2, 0}; // 在设备上
cscVal[nnz]    = { 4.0, 1.0, 8.0, 2.0 }; // 在设备上
size_t pBufferSizeInBytes = 0;
void *pBuffer = NULL;
int *P = NULL;

// 步骤 1：分配缓冲区
mcsparseXcscsort_bufferSizeExt(handle, m, n, nnz, cscColPtr,
↪cscRowInd,
&pBufferSizeInBytes);
mcMalloc( &pBuffer, sizeof(char)* pBufferSizeInBytes);

// 步骤 2：设置置换向量 P 为单位映射
mcMalloc( (void**)&P, sizeof(int)*nnz);
mcsparseCreateIdentityPermutation(handle, nnz, P);

// 步骤 3：对 CSC 格式进行排序
mcsparseXcscsort(handle, m, n, nnz, descrA, cscColPtr, cscRowInd,
↪P, pBuffer);

// 步骤 4：收集排序后的 cscVal
mcsparseDgthr(handle, nnz, cscVal, cscVal_sorted, P, MCSPARSE_
↪INDEX_BASE_ZERO);
```

- 如果 pBuffer != NULL，该例程不需要额外的存储空间
- 如果流式有序内存分配器可用，则该例程支持异步执行

输入

参数	设备或主机	含义
handle	主机	处理 mcSPARSE 库上下文的句柄。
m	主机	矩阵 A 的行数。
n	主机	矩阵 A 的列数。
nnz	主机	矩阵 A 中非零元素的数量。
cscColPtr	设备	整数数组，包含 m+1 个元素，其中每个元素表示每一行的起始位置和最后一行的结束位置加一。
cscRowInd	设备	整数数组，包含矩阵 A 中 nnz 个的未排序列索引。
P	设备	整数数组，包含 nnz 个未排序元素的映射索引。为构建 csrVal，用户需要设置 P=0:1:(nnz-1)。
pBuffer	设备	用户分配的缓冲区；缓冲区的大小由 coosort_bufferSizeExt() 返回。

输出

参数	设备或主机	含义
cscRowInd	设备	整数数组，包含矩阵 A 中 nnz 个已排序的列索引。
P	设备	整数数组，包含 nnz 个已排序的映射索引。
pBufferSizeInBytes	主机	缓冲区的字节数。

有关返回状态的描述，请参见mcsparseStatus_t。

13.15 mcsparseXcsru2csr()

```
mcsparseStatus_t
mcsparseCreateCsru2csrInfo(csru2csrInfo_t *info);

mcsparseStatus_t
mcsparseDestroyCsru2csrInfo(csru2csrInfo_t info);

mcsparseStatus_t
mcsparseScsru2csr_bufferSizeExt(mcsparseHandle_t handle,
                                int              m,
                                int              n,
                                int              nnz,
                                float*           csrVal,
                                const int*       csrRowPtr,
                                int*             csrColInd,
                                csru2csrInfo_t   info,
                                size_t*          pBufferSizeInBytes)

mcsparseStatus_t
mcsparseDcsru2csr_bufferSizeExt(mcsparseHandle_t handle,
                                int              m,
                                int              n,
                                int              nnz,
                                double*          csrVal,
                                const int*       csrRowPtr,
                                int*             csrColInd,
                                csru2csrInfo_t   info,
                                size_t*          pBufferSizeInBytes)

mcsparseStatus_t
```

(下页继续)

(续上页)

```
mcsparseCcsru2csr_bufferSizeExt(mcsparseHandle_t handle,
                                int              m,
                                int              n,
                                int              nnz,
                                mcComplex*       csrVal,
                                const int*       csrRowPtr,
                                int*             csrColInd,
                                csru2csrInfo_t   info,
                                size_t*          pBufferSizeInBytes)

mcsparseStatus_t
mcsparseZcsru2csr_bufferSizeExt(mcsparseHandle_t handle,
                                int              m,
                                int              n,
                                int              nnz,
                                mcDoubleComplex* csrVal,
                                const int*       csrRowPtr,
                                int*             csrColInd,
                                csru2csrInfo_t   info,
                                size_t*          pBufferSizeInBytes)
```

```
mcsparseStatus_t
mcsparseScsru2csr(mcsparseHandle_t         handle,
                  int                      m,
                  int                      n,
                  int                      nnz,
                  const mcsparseMatDescr_t descrA,
                  float*                   csrVal,
                  const int*               csrRowPtr,
                  int*                     csrColInd,
                  csru2csrInfo_t           info,
                  void*                    pBuffer)

mcsparseStatus_t
mcsparseDcsru2csr(mcsparseHandle_t         handle,
                  int                      m,
                  int                      n,
                  int                      nnz,
                  const mcsparseMatDescr_t descrA,
                  double*                  csrVal,
                  const int*               csrRowPtr,
                  int*                     csrColInd,
                  csru2csrInfo_t           info,
                  void*                    pBuffer)

mcsparseStatus_t
mcsparseCcsru2csr(mcsparseHandle_t         handle,
                  int                      m,
                  int                      n,
                  int                      nnz,
                  const mcsparseMatDescr_t descrA,
                  mcComplex*               csrVal,
                  const int*               csrRowPtr,
                  int*                     csrColInd,
                  csru2csrInfo_t           info,
                  void*                    pBuffer)
```

(下页继续)

(续上页)

```
mcsparseStatus_t
mcsparseZcsru2csr(mcsparseHandle_t         handle,
                  int                      m,
                  int                      n,
                  int                      nnz,
                  const mcsparseMatDescr_t descrA,
                  mcDoubleComplex*         csrVal,
                  const int*               csrRowPtr,
                  int*                     csrColInd,
                  csru2csrInfo_t           info,
                  void*                    pBuffer)
```

```
mcsparseStatus_t
mcsparseScsr2csru(mcsparseHandle_t         handle,
                  int                      m,
                  int                      n,
                  int                      nnz,
                  const mcsparseMatDescr_t descrA,
                  float*                   csrVal,
                  const int*               csrRowPtr,
                  int*                     csrColInd,
                  csru2csrInfo_t           info,
                  void*                    pBuffer)

mcsparseStatus_t
mcsparseDcsr2csru(mcsparseHandle_t         handle,
                  int                      m,
                  int                      n,
                  int                      nnz,
                  const mcsparseMatDescr_t descrA,
                  double*                  csrVal,
                  const int*               csrRowPtr,
                  int*                     csrColInd,
                  csru2csrInfo_t           info,
                  void*                    pBuffer)

mcsparseStatus_t
mcsparseCcsr2csru(mcsparseHandle_t         handle,
                  int                      m,
                  int                      n,
                  int                      nnz,
                  const mcsparseMatDescr_t descrA,
                  mcComplex*               csrVal,
                  const int*               csrRowPtr,
                  int*                     csrColInd,
                  csru2csrInfo_t           info,
                  void*                    pBuffer)

mcsparseStatus_t
mcsparseZcsr2csru(mcsparseHandle_t         handle,
                  int                      m,
                  int                      n,
                  int                      nnz,
                  const mcsparseMatDescr_t descrA,
                  mcDoubleComplex*         csrVal,
```

(下页继续)

(续上页)

```
                const int*                  csrRowPtr,
                int*                        csrColInd,
                csru2csrInfo_t              info,
                void*                       pBuffer)
```

此函数将未排序的 CSR 格式转化为 CSR 格式，反之亦然。该操作是原地变换。

此函数是 csrsort 和 gthr 的封装器。

我们需要名为 csru2csrInfo 的不透明结构体来保持排序向量的不可见性。函数 (mcsparseCreateCsru2csrInfo，mcsparseDestroyCsru2csrInfo) 分别用来初始化和销毁不透明结构体。

mcsparse[S|D|C|Z]csru2csr_bufferSizeExt 函数返回缓冲区大小。在 csru2csrInfo 中分配了排列向量 p 的内存。排序向量的生命周期和 csru2csrInfo 的生命周期相同。

mcsparse[S|D|C|Z]csru2csr 函数执行从未排序 CSR 格式到已排序 CSR 格式的正向转换。首先调用 csrsort 生成排序向量 P，随后调用 P 执行转换。

mcsparse[S|D|C|Z]csr2csru 函数执行从已排序 CSR 格式到未排序 CSR 格式的反向转换。P 被用于还原回未排序的格式。

如果 pBuffer != NULL，例程 mcsparse<t>csru2csr() 有以下特性：

- 如果流式有序内存分配器可用，则该例程支持异步执行

如果 pBuffer != NULL，例程 mcsparse<t>csr2csru() 有以下特性：

- 该例程不需要额外的存储空间
- 该例程支持异步执行

下面表格描述了 csr2csru_bufferSizeExt 和 csr2csru 的参数。

输入

参数	设备或主机	含义
handle	主机	处理 mcSPARSE 库上下文的句柄。
m	主机	矩阵 A 的行数。
n	主机	矩阵 A 的列数。
nnz	主机	矩阵 A 的非零元素个数。
descrA	主机	矩阵 A 的描述符。支持的矩阵类型是 MCSPARSE_MATRIX_TYPE_GENERAL，此外，支持的索引基数是 MCSPARSE_INDEX_BASE_ZERO 和 MCSPARSE_INDEX_BASE_ONE。
csrVal	设备	<type> 数组，包含矩阵 A 中 nnz 个未排序非零元素。
csrRowsPtr	设备	由 m+1 个元素组成的整型数组。数组包含了每行的起始位置和最后一行的末尾位置加一。
csrColInd	设备	整数数组，包含矩阵 A 中 nnz 个未排序的列索引。
info	主机	不透明结构体，由 mcsparse CreateCsru2csrInfo() 初始化。
pBuffer	设备	用户分配的缓冲区；大小由 csru2csr_bufferSizeExt() 返回。

输出

参数	设备或主机	含义
csrVal	设备	<type> 数组，存储矩阵 A 中 nnz 个已排序的元素。
csrColInd	设备	整数数组，存储矩阵 A 中 nnz 个已排序的列索引。
pBufferSizeInBytes	主机	缓冲区的字节数。

有关返回状态的描述，请参见mcsparseStatus_t。

13.16 mcsparseXpruneDense2csr()

```
mcsparseStatus_t
mcsparseSpruneDense2csr_bufferSizeExt(mcsparseHandle_t          handle,
                                      int                       m,
                                      int                       n,
                                      const float*              A,
                                      int                       lda,
                                      const float*              threshold,
                                      const mcsparseMatDescr_t  descrC,
                                      const float*              csrValC,
                                      const int*                csrRowPtrC,
                                      const int*                csrColIndC,
                                      size_t*
 ↪pBufferSizeInBytes)

mcsparseStatus_t
mcsparseDpruneDense2csr_bufferSizeExt(mcsparseHandle_t          handle,
                                      int                       m,
                                      int                       n,
                                      const double*             A,
                                      int                       lda,
                                      const double*             threshold,
                                      const mcsparseMatDescr_t  descrC,
                                      const double*             csrValC,
                                      const int*                csrRowPtrC,
                                      const int*                csrColIndC,
                                      size_t*
 ↪pBufferSizeInBytes)
```

```
mcsparseStatus_t
mcsparseSpruneDense2csrNnz(mcsparseHandle_t          handle,
                           int                       m,
                           int                       n,
                           const float*              A,
                           int                       lda,
                           const float*              threshold,
                           const mcsparseMatDescr_t  descrC,
                           int*                      csrRowPtrC,
                           int*                      nnzTotalDevHostPtr,
                           void*                     pBuffer)

mcsparseStatus_t
mcsparseDpruneDense2csrNnz(mcsparseHandle_t          handle,
                           int                       m,
                           int                       n,
                           const double*             A,
                           int                       lda,
                           const double*             threshold,
                           const mcsparseMatDescr_t  descrC,
                           int*                      csrRowPtrC,
                           int*                      nnzTotalDevHostPtr,
                           void*                     pBuffer)
```

```
mcsparseStatus_t
mcsparseSpruneDense2csr(mcsparseHandle_t          handle,
                        int                       m,
                        int                       n,
                        const float*              A,
                        int                       lda,
                        const float*              threshold,
                        const mcsparseMatDescr_t  descrC,
                        float*                    csrValC,
                        const int*                csrRowPtrC,
                        int*                      csrColIndC,
                        void*                     pBuffer)

mcsparseStatus_t
mcsparseDpruneDense2csr(mcsparseHandle_t          handle,
                        int                       m,
                        int                       n,
                        const double*             A,
                        int                       lda,
                        const double*             threshold,
                        const mcsparseMatDescr_t  descrC,
                        double*                   csrValC,
                        const int*                csrRowPtrC,
                        int*                      csrColIndC,
                        void*                     pBuffer)
```

此函数将稠密矩阵精简并转换为 CSR 格式的稀疏矩阵。

此函数通过给定的稠密矩阵 `A` 和非负值 `threshold` 返回稀疏矩阵 `C`。

该实现通过以下操作进行转换：

首先，用户需要分配 `m+1` 个元素的 `csrRowPtrC` 数组，并且使用 `pruneDense2csrNnz` 函数来确定每行非零列的数量。其次，用户通过 `(nnzC=*nnzTotalDevHostPtr)` 函数或 `(nnzC=csrRowPtrC[m]-csrRowPtrC[0])` 函数来收集 `nnzC` (矩阵 `C` 的非零元素个数)，以及分配 `nnzC` 个元素的 `csrValC` 数组和 `nnzC` 个元素的 `csrColIndC` 数组。最后，调用 `pruneDense2csr()` 函数完成转换。

用户必须调用 `pruneDense2csr_bufferSizeExt()` 来获取 `pruneDense2csr()` 所需的缓冲区大小，分配缓冲区，并将缓冲区指针传递给 `pruneDense2csr()`。

例程 `mcsparse<t>pruneDense2csrNnz()` 有以下特性：

- 此函数需要内部分配的临时额外存储空间
- 如果流式有序内存分配器可用，则该例程支持异步执行

例程 `mcsparse<t>DpruneDense2csr()` 有以下特性：

- 该例程不需要额外的存储空间
- 该例程支持异步执行

输入

参数	设备或主机	含义
handle	主机	处理 mcSPARSE 库上下文的句柄。
m	主机	矩阵 A 的行数。
n	主机	矩阵 A 的列数。
A	设备	维度为 (lda，n) 的数组。
lda	设备	矩阵 A 的主维度。主维度至少是 max(1，m)。
threshold	设备或主机	用于丢弃矩阵 A 中元素的阈值 threshold 能够指向设备内存或主机内存。
descrC	主机	矩阵 C 的描述符。支持的矩阵类型是 MCSPARSE_MATRIX_TYPE_GENERAL，此外，支持的索引基数是 MCSPARSE_INDEX_BASE_ZERO 和 MCSPARSE_INDEX_BASE_ONE。
pBuffer	设备	由用户分配的缓冲区：大小由 pruneDense2csr_bufferSizeExt() 返回。

输出

参数	设备或主机	含义
nnzTotalDevHostPtr	设备或主机	矩阵 C 中非零元素的总数。nnzTotalDevHostPtr 指向设备内存或主机内存。
csrValC	设备	<type> 数组，包括矩阵 C 中 nnzC 个中的非零元素。
csrRowsPtrC	设备	由 m+1 个元素组成的整型数组，含有 m+1 个元素。数组包含每行每行的起始位置和最后一行的末尾位置加一。
csrColIndC	设备	整数数组，存储矩阵 C 中的 nnzC 个列索引。
pBufferSizeInBytes	主机	缓冲区的字节数。

有关返回状态的描述，请参见mcsparseStatus_t。

13.17 mcsparseXpruneCsr2csr()

```
mcsparseStatus_t
mcsparseSpruneCsr2csr_bufferSizeExt(mcsparseHandle_t          handle,
                                    int                       m,
                                    int                       n,
                                    int                       nnzA,
                                    const mcsparseMatDescr_t  descrA,
                                    const float*              csrValA,
                                    const int*                csrRowPtrA,
                                    const int*                csrColIndA,
                                    const float*              threshold,
                                    const mcsparseMatDescr_t  descrC,
                                    const float*              csrValC,
                                    const int*                csrRowPtrC,
                                    const int*                csrColIndC,
                                    size_t*                   BufferSize)

mcsparseStatus_t
mcsparseDpruneCsr2csr_bufferSizeExt(mcsparseHandle_t          handle,
                                    int                       m,
                                    int                       n,
```

(下页继续)

(续上页)

```
                                int                         nnzA,
                                const mcsparseMatDescr_t    descrA,
                                const double*               csrValA,
                                const int*                  csrRowPtrA,
                                const int*                  csrColIndA,
                                const double*               threshold,
                                const mcsparseMatDescr_t    descrC,
                                const double*               csrValC,
                                const int*                  csrRowPtrC,
                                const int*                  csrColIndC,
                                size_t*                     BufferSize)
```

```
mcsparseStatus_t
mcsparseSpruneCsr2csrNnz(mcsparseHandle_t         handle,
                         int                      m,
                         int                      n,
                         int                      nnzA,
                         const mcsparseMatDescr_t descrA,
                         const float*             csrValA,
                         const int*               csrRowPtrA,
                         const int*               csrColIndA,
                         const float*             threshold,
                         const mcsparseMatDescr_t descrC,
                         int*                     csrRowPtrC,
                         int*                     nnzTotalDevHostPtr,
                         void*                    pBuffer)

mcsparseStatus_t
mcsparseDpruneCsr2csrNnz(mcsparseHandle_t         handle,
                         int                      m,
                         int                      n,
                         int                      nnzA,
                         const mcsparseMatDescr_t descrA,
                         const double*            csrValA,
                         const int*               csrRowPtrA,
                         const int*               csrColIndA,
                         const double*            threshold,
                         const mcsparseMatDescr_t descrC,
                         int*                     csrRowPtrC,
                         int*                     nnzTotalDevHostPtr,
                         void*                    pBuffer)
```

```
mcsparseStatus_t
mcsparseSpruneCsr2csr(mcsparseHandle_t          handle,
                      int                       m,
                      int                       n,
                      int                       nnzA,
                      const mcsparseMatDescr_t  descrA,
                      const float*              csrValA,
                      const int*                csrRowPtrA,
                      const int*                csrColIndA,
                      const float*              threshold,
                      const mcsparseMatDescr_t  descrC,
                      float*                    csrValC,
                      const int*                csrRowPtrC,
                      int*                      csrColIndC,
```

(下页继续)

(续上页)

```
                          void*                       pBuffer)

mcsparseStatus_t
mcsparseDpruneCsr2csr(mcsparseHandle_t         handle,
                          int                         m,
                          int                         n,
                          int                         nnzA,
                          const mcsparseMatDescr_t descrA,
                          const double*               csrValA,
                          const int*                  csrRowPtrA,
                          const int*                  csrColIndA,
                          const double*               threshold,
                          const mcsparseMatDescr_t descrC,
                          double*                     csrValC,
                          const int*                  csrRowPtrC,
                          int*                        csrColIndC,
                          void*                       pBuffer)
```

此函数将稠密矩阵精简并转换为 CSR 格式的稀疏矩阵。

该实现通过以下操作进行转换：

首先，用户需要分配 m+1 个元素的 csrRowPtrC 数组，并且使用 pruneCsr2csrNnz() 函数来确定每行非零列的数量。其次，用户通过 (nnzC=*nnzTotalDevHostPtr) 函数或 (nnzC=csrRowPtrC[m]-csrRowPtrC[0]) 函数来收集 nnzC (矩阵 C 的非零元素个数)，并分配 nnzC 个元素的 csrValC 数组和 nnzC 个元素的 csrColIndC 数组。最后，调用 pruneDense2csr() 函数完成转换。

用户必须调用 pruneCsr2csr_bufferSizeExt() 来获取 pruneCsr2csr() 需求的缓冲区大小，然后分配缓冲区并将缓冲指针传递给 pruneCsr2csr()。

例程 mcsparse<t>pruneCsr2csrNnz() 有以下特性：

- 函数需要额外分配内部临时存储
- 如果流序内存分配器可用，则例程支持异步执行

例程 mcsparse<t>pruneCsr2csr() 有以下特性：

- 例程不需要额外存储
- 例程支持异步执行

输入

参数	设备或主机	含义
handle	主机	处理 mcSPARSE 库上下文的句柄。
m	主机	矩阵 A 的行数。
n	主机	矩阵 A 的列数。
nnzA	主机	矩阵 A 中非零元素的个数。
descrA	主机	矩阵 A 的描述符。支持的矩阵类型是 MCSPARSE_MATRIX_TYPE_GENERAL，此外，支持的索引基数是 MCSPARSE_INDEX_BASE_ZERO 和 MCSPARSE_INDEX_BASE_ONE。
csrValA	设备	<type> 数组，存储矩阵 A 中非零元素的个数。
csrRowsPtrA	设备	整数数组，含有 m+1 个元素数组包含了每行的起始位置和最后一行的末尾位置加一。
csrColIndA	设备	整数数组，存储矩阵 A 中 nnzA 个列索引。
threshold	设备或主机	用于丢弃矩阵 A 中元素的阈值。threshold 指向设备内存或主机内存。
descrC	主机	矩阵 C 的描述符。支持的矩阵类型是 MCSPARSE_MATRIX_TYPE_GENERAL，此外，支持的索引基数是 MCSPARSE_INDEX_BASE_ZERO 和 MCSPARSE_INDEX_BASE_ONE。
pBuffer	设备	用户分配的缓冲区；大小由 pruneCsr2csr_bufferSizeExt() 返回。

输出

参数	设备或主机	含义
nnzTotalDevHostPtr	设备或主机	矩阵 C 中非零元素的个数。nnzTotalDevHostPtr 指向设备内存或主机内存。
csrValC	设备	<type> 数组，存储 nnC 个矩阵 C 中的非零元素个数。
csrRowsPtrC	设备	整数数组，含有 m+1 个元素。数组包含了每行的起始位置和最后一行的末尾位置加一。
csrColIndC	设备	整数数组，存储矩阵 C 中 nnzC 个的列索引。
BufferSize	主机	缓冲区的字节数。

有关返回状态的描述，请参见mcsparseStatus_t。

13.18 mcsparseXpruneDense2csrPercentage()

```
mcsparseStatus_t
mcsparseSpruneDense2csrByPercentage_bufferSizeExt
                  (mcsparseHandle_t          handle,
                   int                       m,
                   int                       n,
                   const float*              A,
                   int                       lda,
                   float                     percentage,
                   const mcsparseMatDescr_t  descrC,
                   const float*              csrValC,
                   const int*                csrRowPtrC,
                   const int*                csrColIndC,
                   pruneInfo_t               info,
                   size_t*                   BufferSize)

mcsparseStatus_t
```

(下页继续)

(续上页)

```
mcsparseDpruneDense2csrByPercentage_bufferSizeExt
                     (mcsparseHandle_t         handle,
                      int                      m,
                      int                      n,
                      const double*            A,
                      int                      lda,
                      float                    percentage,
                      const mcsparseMatDescr_t descrC,
                      const double*            csrValC,
                      const int*               csrRowPtrC,
                      const int*               csrColIndC,
                      pruneInfo_t              info,
                      size_t*                  BufferSize)
```

```
mcsparseStatus_t
mcsparseSpruneDense2csrNnzByPercentage(mcsparseHandle_t         handle,
                                       int                      m,
                                       int                      n,
                                       const float*             A,
                                       int                      lda,
                                       float                    percentage,
                                       const mcsparseMatDescr_t descrC,
                                       int*                     csrRowPtrC,
                                       int*
 ↪nnzTotalDevHostPtr,
                                       pruneInfo_t              info,
                                       void*                    pBuffer)

mcsparseStatus_t
mcsparseDpruneDense2csrNnzByPercentage(mcsparseHandle_t         handle,
                                       int                      m,
                                       int                      n,
                                       const double*            A,
                                       int                      lda,
                                       float                    percentage,
                                       const mcsparseMatDescr_t descrC,
                                       int*                     csrRowPtrC,
                                       int*
 ↪nnzTotalDevHostPtr,
                                       pruneInfo_t              info,
                                       void*                    pBuffer)
```

```
mcsparseStatus_t
mcsparseSpruneDense2csrByPercentage(mcsparseHandle_t         handle,
                                    int                      m,
                                    int                      n,
                                    const float*             A,
                                    int                      lda,
                                    float                    percentage,
                                    const mcsparseMatDescr_t descrC,
                                    float*                   csrValC,
                                    const int*               csrRowPtrC,
                                    int*                     csrColIndC,
                                    pruneInfo_t              info,
                                    void*                    pBuffer)
```

(下页继续)

(续上页)

```
mcsparseStatus_t
mcsparseDpruneDense2csrByPercentage(mcsparseHandle_t          handle,
                                    int                       m,
                                    int                       n,
                                    const double*             A,
                                    int                       lda,
                                    float                     percentage,
                                    const mcsparseMatDescr_t  descrC,
                                    double*                   csrValC,
                                    const int*                csrRowPtrC,
                                    int*                      csrColIndC,
                                    pruneInfo_t               info,
                                    void*                     pBuffer)
```

此函数通过百分比将稠密矩阵精简为稀疏矩阵。

给定的稠密矩阵 A 和非负值 threshold，此函数通过以下三个步骤计算稀疏矩阵 C：

步骤 1：按照绝对值对矩阵 A 进行升序排序。

步骤 2：根据参数 percentage 选择阈值

步骤 3：使用参数 threshold 调用 pruneDense2csr()。

该实现通过以下操作进行转换：

首先，用户需要分配 m+1 个元素的 csrRowPtrC 数组，并且使用 pruneDense2csrNnzByPercentage() 函数来确定每行非零列的数量。其次，用户通过 (nnzC=*nnzTotalDevHostPtr) 函数或 (nnzC=csrRowPtrC[m]-csrRowPtrC[0]) 函数来收集 nnzC (矩阵 C 的非零元素个数)，并分配 nnzC 个元素的 csrValC 数组和 nnzC 个整数元素的 csrColIndC 数组。最后，调用 pruneDense2csrByPercentage() 函数完成转换。

用户必须通过调用 pruneDense2csrByPercentage_bufferSizeExt() 来获取 pruneDense2csrByPercentage() 需求的缓冲区大小，然后分配缓冲区，并将缓冲指针传递给 pruneDense2csrByPercentage()。

注解:

- percentage 值必须不大于 100。否则，将返回 MCSPARSE_STATUS_INVALID_VALUE。
- 和 pruneCsr2csrByPercentage() 函数不同的地方在于矩阵 A 中的零元素不会被忽略，包括零在内的所有元素都参与排序。

例程 mcsparse<t>pruneDense2csrNnzByPercentage() 有以下特性：

- 函数需要额外分配内部临时存储
- 如果流序内存分配器可用，则例程支持异步执行

例程 mcsparse<t>pruneDense2csrByPercentage() 有以下特性：

- 例程无需额外存储
- 例程支持异步执行

输出

参数	设备或主机	含义
handle	主机	处理 mcSPARSE 库上下文的句柄。
m	主机	矩阵 A 的行数。
n	主机	矩阵 A 的列数。
A	设备	维度为 (lda，n) 的数组。
lda	设备	矩阵 A 的主维度。主维度至少是 max(1，m)。
percentage	主机	percentage <=100 和 percentage >= 0
descrC	主机	矩阵 C 的描述符。支持的矩阵类型是 MCSPARSE_MATRIX_TYPE_GENERAL，此外，支持的索引基数是 MCSPARSE_INDEX_BASE_ZERO 和 MCSPARSE_INDEX_BASE_ONE。
pBuffer	设备	由用户分配的缓冲区；大小由 pruneDense2csrByPercentage_bufferSizeExt() 返回。

输出

参数	设备或主机	含义
nnzTotalDevHostPtr	设备或主机	矩阵 C 中非零元素的个数。nnzTotalDevHostPtr 指向设备内存或主机内存
csrValC	设备	<type> 数组，存储矩阵 C 中的 nnzC 个非零元素。
csrRowsPtrC	设备	整数数组，含有 m+1 个元素。数组包含了每行的起始位置和最后一行的末尾位置加一。
csrColIndC	设备	整数数组，包含矩阵 C 中 nnzC 个中的列索引
BufferSize	主机	缓冲区的字节数。

有关返回状态的描述，请参见mcsparseStatus_t。

13.19 mcsparseXpruneCsr2csrByPercentage()

```
mcsparseStatus_t
mcsparseSpruneCsr2csrByPercentage_bufferSizeExt(mcsparseHandle_t
 ↪handle,
                                                 int                         m,
                                                 int                         n,
                                                 int
 ↪nnzA,
                                                 const mcsparseMatDescr_t
 ↪descrA,
                                                 const float*
 ↪csrValA,
                                                 const int*
 ↪csrRowPtrA,
                                                 const int*
 ↪csrColIndA,
                                                 float
 ↪percentage,
                                                 const mcsparseMatDescr_t
 ↪descrC,
                                                 const float*
 ↪csrValC,
                                                 const int*
 ↪csrRowPtrC,
```

(下页继续)

(续上页)

```
                                                const int*
↪csrColIndC,
                                                pruneInfo_t
↪info,
                                                size_t*
↪BufferSize)

mcsparseStatus_t
mcsparseDpruneCsr2csrByPercentage_bufferSizeExt(mcsparseHandle_t
↪handle,
                                                int                   m,
                                                int                   n,
                                                int
↪nnzA,
                                                const mcsparseMatDescr_t
↪descrA,
                                                const double*
↪csrValA,
                                                const int*
↪csrRowPtrA,
                                                const int*
↪csrColIndA,
                                                float
↪percentage,
                                                const mcsparseMatDescr_t
↪descrC,
                                                const double*
↪csrValC,
                                                const int*
↪csrRowPtrC,
                                                const int*
↪csrColIndC,
                                                pruneInfo_t
↪info,
                                                size_t*
↪BufferSize)

mcsparseStatus_t
mcsparseSpruneCsr2csrNnzByPercentage(mcsparseHandle_t        handle,
                                     int                     m,
                                     int                     n,
                                     int                     nnzA,
                                     const mcsparseMatDescr_t descrA,
                                     const float*            csrValA,
                                     const int*              csrRowPtrA,
                                     const int*              csrColIndA,
                                     float                   percentage,
                                     const mcsparseMatDescr_t descrC,
                                     int*                    csrRowPtrC,
                                     int*
↪nnzTotalDevHostPtr,
                                     pruneInfo_t             info,
                                     void*                   pBuffer)

mcsparseStatus_t
mcsparseDpruneCsr2csrNnzByPercentage(mcsparseHandle_t        handle,
```

(下页继续)

(续上页)

```
                                    int                         m,
                                    int                         n,
                                    int                         nnzA,
                                    const mcsparseMatDescr_t    descrA,
                                    const double*               csrValA,
                                    const int*                  csrRowPtrA,
                                    const int*                  csrColIndA,
                                    float                       percentage,
                                    const mcsparseMatDescr_t    descrC,
                                    int*                        csrRowPtrC,
                                    int*
↪nnzTotalDevHostPtr,
                                    pruneInfo_t                 info,
                                    void*                       pBuffer)
```

```
mcsparseStatus_t
mcsparseSpruneCsr2csrByPercentage(mcsparseHandle_t          handle,
                                    int                         m,
                                    int                         n,
                                    int                         nnzA,
                                    const mcsparseMatDescr_t    descrA,
                                    const float*                csrValA,
                                    const int*                  csrRowPtrA,
                                    const int*                  csrColIndA,
                                    float                       percentage,
                                    const mcsparseMatDescr_t    descrC,
                                    float*                      csrValC,
                                    const int*                  csrRowPtrC,
                                    int*                        csrColIndC,
                                    pruneInfo_t                 info,
                                    void*                       pBuffer)

mcsparseStatus_t
mcsparseDpruneCsr2csrByPercentage(mcsparseHandle_t          handle,
                                    int                         m,
                                    int                         n,
                                    int                         nnzA,
                                    const mcsparseMatDescr_t    descrA,
                                    const double*               csrValA,
                                    const int*                  csrRowPtrA,
                                    const int*                  csrColIndA,
                                    float                       percentage,
                                    const mcsparseMatDescr_t    descrC,
                                    double*                     csrValC,
                                    const int*                  csrRowPtrC,
                                    int*                        csrColIndC,
                                    pruneInfo_t                 info,
                                    void*                       pBuffer)
```

此函数通过百分比将稠密矩阵精简为稀疏矩阵。

给定的稠密矩阵 `A` 和非负值 `percentage`，此函数通过以下三个步骤计算稀疏矩阵 `C`：

步骤 1：按照绝对值对矩阵 `A` 进行升序排序。

步骤 2：根据参数 `percentage` 选择阈值 (`threshold`)。

步骤 3：使用参数 `threshold` 调用 `pruneCsr2csr()`。

该实现通过以下操作进行转换：

首先，用户需要分配 m+1 个元素的 csrRowPtrC 数组，并且使用 pruneCsr2csrNnzByPercentage() 函数来确定每行非零列的数量。其次，用户通过 (nnzC=*nnzTotalDevHostPtr) 函数或 (nnzC=csrRowPtrC[m]-csrRowPtrC[0]) 函数来收集 nnzC (矩阵 C 的非零元素个数)，以及分配 nnzC 个元素的 csrValC 数组和 nnzC 个整数元素的 csrColIndC 数组最后，调用 pruneCsr2csrByPercentage() 函数完成转换。

用户必须调用 pruneCsr2csrByPercentage_bufferSizeExt() 来获取 pruneCsr2csrByPercentage() 所需的缓冲区大小，分配缓冲区，并将缓冲区指针传递给 pruneCsr2csrByPercentage()。

注解： percentage 值必须不大于 100。否则，将返回 MCSPARSE_STATUS_INVALID_VALUE。

例程 mcsparse<t>pruneCsr2csrNnzByPercentage() 有以下特性：

- 函数需要额外分配内部临时存储
- 如果流序内存分配器可用，例程支持异步执行

例程 mcsparse<t>pruneCsr2csrByPercentage() 有以下特性：

- 例程不需要额外存储
- 例程支持异步执行

输入

参数	设备或主机	含义
handle	主机	处理 mcSPARSE 库上下文的句柄。
m	主机	矩阵 A 的行数。
n	主机	矩阵 A 的列数。
nnzA	主机	矩阵 A 的非零数。
descrA	主机	矩阵 A 的描述符。支持的矩阵类型为 MCSPARSE_MATRIX_TYPE_GENERAL，同时，支持的索引基也有 MCSPARSE_INDEX_BASE_ZERO 和 MCSPARSE_INDEX_BASE_ONE
csrValA	设备	类型为 <type> 的数组，包含矩阵 A 中 nnzA 个非零元素。
csrRowsPtrA	设备	由 m+1 个元素组成的整型数组，其中包含每一行的起始位置和最后一行的结束位置加 1。
csrColIndA	设备	整型数组包含矩阵 A 中 nnzA 个的列索引。
percentage	主机	percentage <=100 和 percentage >= 0
descrC	主机	矩阵 C 的描述符。支持的矩阵类型为 MCSPARSE_MATRIX_TYPE_GENERAL，同时，支持的索引基也有 MCSPARSE_INDEX_BASE_ZERO 和 MCSPARSE_INDEX_BASE_ONE
pBuffer	设备	用户分配的缓冲区；其大小由 pruneCsr2csrByPercentage_bufferSizeExt() 返回。

输出

parameter	设备或主机	含义
nnzTotalDevHostPtr	设备或主机	矩阵 C 的非零元素总数。nnzTotalDevHostPtr 可以指向设备内存或主机内存。
csrValC	设备	类型为 <type> 的数组，包含矩阵 C 中 nnzC 个非零元素。
csrRowsPtrC	设备	由 m+1 个元素组成的整型数组，其中包含每一行的起始位置和最后一行的结束位置加 1。
csrColIndC	设备	整型数组包含矩阵 C 中 nnzC 个列索引
BufferSize	主机	缓冲区的字节数。

有关返回状态的描述，请参见mcsparseStatus_t。

13.20 mcsparse<t>nnz_compress()

```
mcsparseStatus_t
mcsparseSnnz_compress(mcsparseHandle_t         handle,
                      int                      m,
                      const mcsparseMatDescr_t descr,
                      const float*             csrValA,
                      const int*               csrRowPtrA,
                      int*                     nnzPerRow,
                      int*                     nnzC,
                      float                    tol)

mcsparseStatus_t
mcsparseDnnz_compress(mcsparseHandle_t         handle,
                      int                      m,
                      const mcsparseMatDescr_t descr,
                      const double*            csrValA,
                      const int*               csrRowPtrA,
                      int*                     nnzPerRow,
                      int*                     nnzC,
                      double                   tol)

mcsparseStatus_t
mcsparseCnnz_compress(mcsparseHandle_t         handle,
                      int                      m,
                      const mcsparseMatDescr_t descr,
                      const mcComplex*         csrValA,
                      const int*               csrRowPtrA,
                      int*                     nnzPerRow,
                      int*                     nnzC,
                      mcComplex                tol)

mcsparseStatus_t
mcsparseZnnz_compress(mcsparseHandle_t         handle,
                      int                      m,
                      const mcsparseMatDescr_t descr,
                      const mcDoubleComplex*   csrValA,
                      const int*               csrRowPtrA,
                      int*                     nnzPerRow,
                      int*                     nnzC,
                      mcDoubleComplex          tol)
```

此函数是从 CSR 格式转换为压缩 CSR 格式的第一步。

对于 `mcComplex` 和 `mcDoubleComplex` 情况的一个关键假设是，容差是以实部形式给出的。例如，`tol = 1e-8 + 0*i`，我们提取实部，也就是这个结构体的 x 分量。

- 此函数需要临时的额外内部存储空间。
- 如果存在流式有序内存分配器，此例程支持异步执行。

输入

handle	处理 mcSPARSE 库上下文的句柄
m	矩阵 A 的行数。
descrA	矩阵 A 的描述符。支持的矩阵类型是 MCSPARSE_MATRIX_TYPE_GENERAL。同时，支持的基准索引是 MCSPARSE_INDEX_BASE_ZERO 和 MCSPARSE_INDEX_BASE_ONE。
csrValA	CSR 非压缩值数组
csrRowPtrA	相应的输入非压缩行指针。
tol	非负容差，用于确定一个数是否小于等于它。

输出

nnzPerRow	这个数组包含每行绝对值大于 tol 的元素数量
nnzC	总共有多少个绝对值大于 tol 的元素的主机/设备指针。

有关返回状态的描述，请参见mcsparseStatus_t。

14 mcSPARSE 通用 API 参考

mcSPARSE 通用 API 允许以灵活的方式进行最常见的稀疏线性代数运算，例如稀疏矩阵-向量乘法 (SpMV) 和稀疏矩阵-矩阵乘法 (SpMM)。这些新的 API 具有以下功能和特点：

- 设置矩阵的数据布局、批次数和存储格式 (例如 CSR、COO 等)
- 设置输入/输出/计算数据类型。这还允许混合数据类型计算。
- 设置稀疏矩阵索引的类型。
- 选择计算的算法。
- 提供用于内部操作的外部设备内存。
- 对于给定的例程，提供广泛的输入矩阵和向量的一致性检查。这包括验证矩阵的大小、数据类型、布局、允许的操作等。

14.1 通用类型参考

本章节描述了 mcSPARSE 通用类型的参考。

14.1.1 macaDataType_t

本节描述了多个 MXMACA 库共享并在头文件 `library_types.h` 中定义的类型。其中 `macaDataType` 类型是一个用于指定数据的精度枚举类型。当数据引用本身不携带类型（例如 `void*` 类型）时，就会使用它。例如，在例程 `mcsparseSpMM()` 中使用了这个类型。

值	含义	数据类型	头文件
`MACA_R_32F`	数据类型是 32 位 IEEE-754 浮点数	`float`	
`MACA_C_32F`	数据类型是 32 位 IEEE-754 复数浮点数	`mcComplex`	mcComplex.h
`MACA_R_64F`	数据类型是 64 位 IEEE-754 浮点数点数	`double`	
`MACA_C_64F`	数据类型是 64 位 IEEE-754 复数浮点数	`mcDouble Complex`	mcComplex.h
`MACA_R_8I`	数据类型是 8 位整数	`int8_t`	stdint.h
`MACA_R_32I`	数据类型是 32 位整数	`int32_t`	stdint.h

重要提示：

通用 API 例程仅允许在有 原生支持的 GPU 架构上使用文档相应章节中报告的所有数据类型。如果特定的 GPU 型号没有提供对给定数据类型的 原生支持，则例程将返回 `MCSPARSE_STATUS_ARCH_MISMATCH` 错误。

14.1.2 mcsparseFormat_t

此类型指示稀疏矩阵的格式。

值	含义
MCSPARSE_FORMAT_COO	矩阵以坐标格式 (COO) 存储，并采用结构数组 (SoA) 布局。
MCSPARSE_FORMAT_COO_AOS	矩阵以坐标格式 (COO) 存储，并采用结构数组 (SoA) 布局。
MCSPARSE_FORMAT_CSR	矩阵以压缩稀疏行 (CSR) 格式存储。
MCSPARSE_FORMAT_CSC	矩阵以压缩稀疏列 (CSC) 格式存储。
MCSPARSE_FORMAT_BLOCKED_ELL	矩阵以块-ELL(Blocked-Ell) 格式存储。

14.1.3 mcsparseOrder_t

此类型表示稠密矩阵的内存布局。

相关值	含义
MCSPARSE_ORDER_ROW	矩阵按行主序存储。
MCSPARSE_ORDER_COL	矩阵按列主序存储。

14.1.4 mcsparseIndexType_t

用于表示稀疏矩阵索引的索引类型。

相关值	含义
MCSPARSE_INDEX_16U	16 位无符号整数。[1, 65535]
MCSPARSE_INDEX_32I	32 位有符号整数。[1, 2^31 - 1]
MCSPARSE_INDEX_64I	64 位有符号整数。[1, 2^63 - 1]

14.2 稀疏向量 API

该部分描述了用于稀疏向量描述符的 mcSPARSE 辅助函数。

14.2.1 mcsparseCreateSpVec()

```
mcsparseStatus_t
mcsparseCreateSpVec(mcsparseSpVecDescr_t* spVecDescr,
                    int64_t               size,
                    int64_t               nnz,
                    void*                 indices,
                    void*                 values,
                    mcsparseIndexType_t   idxType,
                    mcsparseIndexBase_t   idxBase,
                    macaDataType          valueType)
```

此函数用于初始化稀疏矩阵描述符 spVecDescr。

参数	内存	输入/输出	含义
spVecDescr	主机	输出	稀疏向量描述符
size	主机	输入	稀疏向量的大小
nnz	主机	输入	稀疏向量中非零条目的数量
indices	设备	输入	稀疏向量的索引数组大小为 nnz
values	设备	输入	稀疏向量的索引数组大小为 nnz
idxType	主机	输入	枚举器，用于指定 indices 的数据类型
idxBase	主机	输入	枚举器，用于指定 indices 的基索引
valueType	主机	输入	枚举器，用于指定 values 的数据类型

注解： 如果描述符不会被用作例程的输出参数 (例如转换函数)，则可以安全地对输入指针进行去除 const 属性性变换 (使用 const_cast) 。

有关返回状态的描述，请参见mcsparseStatus_t。

14.2.2 mcsparseDestroySpVec()

```
mcsparseStatus_t
mcsparseDestroySpVec(mcsparseSpVecDescr_t spVecDescr)
```

此函数释放为稀疏向量描述符 spVecDescr 分配的主机内存。

参数	内存	输入/输出	含义
spVecDescr	主机	输入	稀疏向量描述符

有关返回状态的描述，请参见mcsparseStatus_t。

14.2.3 mcsparseSpVecGet()

```
mcsparseStatus_t
mcsparseSpVecGet(mcsparseSpVecDescr_t spVecDescr,
                 int64_t*             size,
                 int64_t*             nnz,
                 void**               indices,
                 void**               values,
                 mcsparseIndexType_t* idxType,
                 mcsparseIndexBase_t* idxBase,
                 macaDataType*        valueType)
```

此函数返回稀疏向量描述符 spVecDescr 的字段。

参数	内存	输入/输出	含义
spVecDescr	主机	输入	稀疏向量描述符
size	主机	输出	稀疏向量的大小
nnz	主机	输出	稀疏向量的非零元素个数
indices	设备	输出	稀疏向量的索引。大小为 nnz 的数组
values	设备	输出	稀疏向量的值。大小为 nnz 的数组
idxType	主机	输出	枚举器指定 indices 的数据类型
idxBase	主机	输出	枚举器指定 indices 的基索引
valueType	主机	输出	枚举器指定 values 的数据类型

有关返回状态的描述，请参见mcsparseStatus_t。

14.2.4 mcsparseSpVecGetIndexBase()

```
mcsparseStatus_t
mcsparseSpVecGetIndexBase(mcsparseSpVecDescr_t spVecDescr,
                          mcsparseIndexBase_t* idxBase)
```

此函数返回稀疏向量描述符 spVecDescr 的 idxBase 字段。

参数	内存	输入/输出	含义
spVecDescr	主机	输入	稀疏向量描述符
idxBase	主机	输出	枚举器，指定 indices 的基准索引

有关返回状态的描述，请参见mcsparseStatus_t。

14.2.5 mcsparseSpVecGetValues()

```
mcsparseStatus_t
mcsparseSpVecGetValues(mcsparseSpVecDescr_t spVecDescr,
                       void**               values)
```

此函数返回稀疏向量描述符 spVecDescr 的 values 字段。

参数	内存	输入/输出	含义
spVecDescr	主机	输入	稀疏向量描述符 descriptor
values	设备	输出	稀疏向量的值。大小为 nnz 的数组

有关返回状态的描述，请参见mcsparseStatus_t。

14.2.6 mcsparseSpVecSetValues()

```
mcsparseStatus_t
mcsparseSpVecSetValues(mcsparseSpVecDescr_t spVecDescr,
                       void*                values)
```

此函数设置稀疏向量描述符 spVecDescr 的 values 字段。

参数	内存	输入/输出	含义
spVecDescr	主机	输入	稀疏向量描述符 descriptor
values	设备	输入	稀疏向量的值。大小为 nnz 的数组

有关返回状态的描述，请参见mcsparseStatus_t。

14.3 稀疏矩阵 API

本节介绍用于稀疏矩阵描述符的 mcSPARSE 辅助函数。

14.3.1 mcsparseCreateCoo()

```
mcsparseStatus_t
mcsparseCreateCoo(mcsparseSpMatDescr_t* spMatDescr,
                  int64_t               rows,
                  int64_t               cols,
                  int64_t               nnz,
                  void*                 cooRowInd,
                  void*                 cooColInd,
                  void*                 cooValues,
                  mcsparseIndexType_t   cooIdxType,
                  mcsparseIndexBase_t   idxBase,
                  macaDataType          valueType)
```

此函数以 COO 格式 (数组结构布局) 初始化稀疏矩阵描述符 spMatDescr。

参数	内存	输入/输出	含义
spMatDescr	主机	输出	稀疏矩阵描述符
rows	主机	输入	稀疏矩阵的行数
cols	主机	输入	稀疏矩阵的列数
nnz	主机	输入	稀疏矩阵的非零元素个数
cooRowInd	设备	输入	稀疏矩阵的行索引。大小为 nnz 的数组
cooColInd	设备	输入	稀疏矩阵的列索引。大小为 nnz 的数组
cooValues	设备	输入	稀疏矩阵的值。大小为 nnz 的数组
cooIdxType	主机	输入	cooRowInd 和 cooColInd 的数据类型
idxBase	主机	输入	cooRowInd 和 cooColInd 的基准索引
valueType	主机	输入	cooValues 的数据类型

注解： 如果描述符不会被用作例程的输出参数 (如转换函数)，则可以安全地去除输入指针的 const 属性 (使用 const_cast)。

有关返回状态的描述，请参见mcsparseStatus_t。

14.3.2 mcsparseCreateCsr()

```
mcsparseStatus_t
mcsparseCreateCsr(mcsparseSpMatDescr_t* spMatDescr,
                  int64_t               rows,
                  int64_t               cols,
                  int64_t               nnz,
                  void*                 csrRowOffsets,
                  void*                 csrColInd,
                  void*                 csrValues,
                  mcsparseIndexType_t   csrRowOffsetsType,
                  mcsparseIndexType_t   csrColIndType,
                  mcsparseIndexBase_t   idxBase,
                  macaDataType          valueType)
```

此函数以 CSR 格式初始化稀疏矩阵的描述符 spMatDescr。

参数	内存	输入/输出	含义
spMatDescr	主机	输出	稀疏矩阵描述符。
rows	主机	输入	稀疏矩阵的行数
cols	主机	输入	稀疏矩阵的列数
nnz	主机	输入	稀疏矩阵的非零元素数量
csrRowOffsets	设备	输入	稀疏矩阵的行偏移量。大小为 rows + 1 的数组。
csrColInd	设备	输入	稀疏矩阵的列索引。大小为 nnz 的数组。
csrValues	设备	输入	稀疏矩阵的值。大小为 nnz 的数组。
csrRowOffsetsType	主机	输入	数据类型为 csrRowOffsets
csrColIndType	主机	输入	数据类型为 csrColInd
idxBase	主机	输入	基础索引的类型为 csrRowOffsets 和 csrColInd
valueType	主机	输入	数据类型为 csrValues

注解： 如果描述符不会用作程序的输出参数 (例如转换函数)，则可以安全地去除输入指针的常量性 (使用 const_cast)。

有关返回状态的描述，请参见mcsparseStatus_t。

14.3.3 mcsparseCreateCsc()

```
mcsparseStatus_t
mcsparseCreateCsc(mcsparseSpMatDescr_t* spMatDescr,
                  int64_t               rows,
                  int64_t               cols,
                  int64_t               nnz,
                  void*                 cscColOffsets,
                  void*                 cscRowInd,
                  void*                 cscValues,
                  mcsparseIndexType_t   cscColOffsetsType,
                  mcsparseIndexType_t   cscRowIndType,
                  mcsparseIndexBase_t   idxBase,
                  macaDataType          valueType)
```

此函数用于初始化用 CSC 格式表示的稀疏矩阵描述符 spMatDescr。

参数	内存	输入/输出	含义
spMatDescr	主机	输出	稀疏矩阵描述符
rows	主机	输入	稀疏矩阵的行数
cols	主机	输入	稀疏矩阵的列数
nnz	主机	输入	稀疏矩阵的非零项数
cscColOffsets	设备	输入	稀疏矩阵的列偏移。大小为 cols + 1 的数组
cscRowInd	设备	输入	稀疏矩阵的行索引。大小为 nnz 的数组
cscValues	设备	输入	稀疏矩阵的值。大小为 nnz 的数组
cscColOffsetsType	主机	输入	cscColOffsets 的数据类
cscRowIndType	主机	输入	cscRowInd 的数据类型
idxBase	主机	输入	cscColOffsets 和 cscRowInd 基准索引
valueType	主机	输入	cscValues 的数据类型

注解: 如果描述符不会被用作例程的输出参数 (例如转换函数)，则可以安全地使用 const_cast 来移除输入指针的 const 限定。

有关返回状态的描述，请参见mcsparseStatus_t。

14.3.4 mcsparseDestroySpMat()

```
mcsparseStatus_t
mcsparseDestroySpMat(mcsparseSpMatDescr_t spMatDescr)
```

此函数释放分配给稀疏矩阵描述符 spMatDescr 的内存。

参数	内存	输入/输出	含义
spMatDescr	主机	输入	稀疏矩阵描述符

有关返回状态的描述，请参见mcsparseStatus_t。

14.3.5 mcsparseCooGet()

```
mcsparseStatus_t
mcsparseCooGet(mcsparseSpMatDescr_t spMatDescr,
               int64_t*             rows,
               int64_t*             cols,
               int64_t*             nnz,
               void**               cooRowInd,
               void**               cooColInd,
               void**               cooValues,
               mcsparseIndexType_t* idxType,
               mcsparseIndexBase_t* idxBase,
               macaDataType*        valueType)
```

此函数返回以 COO 格式 (结构数组布局) 存储的稀疏矩阵描述符 spMatDescr 的字段。

参数	内存	输入/输出	含义
spMatDescr	主机	输入	稀疏矩阵描述符
rows	主机	输出	稀疏矩阵的行数
cols	主机	输出	稀疏矩阵的列数
nnz	主机	输出	稀疏矩阵的非零元素数量
cooRowInd	设备	输出	稀疏矩阵的行索引。大小为 nnz 的数组
cooColInd	设备	输出	稀疏矩阵的列索引。大小为 nnz 的数组
cooValues	设备	输出	稀疏矩阵的非零元素大小为 nnz 的数组
cooIdxType	主机	输出	cooRowInd 和 cooColInd 的数据类型
idxBase	主机	输出	cooRowInd 和 cooColInd 的基准索引
valueType	主机	输出	cooValues 数据类型

有关返回状态的描述，请参见mcsparseStatus_t。

14.3.6 mcsparseCsrGet()

```
mcsparseStatus_t
mcsparseCsrGet(mcsparseSpMatDescr_t spMatDescr,
               int64_t*             rows,
               int64_t*             cols,
               int64_t*             nnz,
               void**               csrRowOffsets,
               void**               csrColInd,
               void**               csrValues,
               mcsparseIndexType_t* csrRowOffsetsType,
               mcsparseIndexType_t* csrColIndType,
               mcsparseIndexBase_t* idxBase,
               macaDataType*        valueType)
```

此函数返回以 CSR 格式存储的稀疏矩阵描述符的 spMatDescr 字段。

参数	内存	输入/输出	含义
spMatDescr	主机	输入	稀疏矩阵描述符
rows	主机	输出	稀疏矩阵的行数
cols	主机	输出	稀疏矩阵的列数
nnz	主机	输出	稀疏矩阵的非零元素数量
csrRowOffsets	设备	输出	稀疏矩阵的行偏移。大小为 rows + 1 的数组
csrColInd	设备	输出	稀疏矩阵的列索引。大小为 nnz 的数组
csrValues	设备	输出	稀疏矩阵的非零元素 nnz
csrRowOffsetsType	主机	输出	csrRowOffsets 的数据类型
csrColIndType	主机	输出	csrColInd 的数据类型
idxBase	主机	输出	csrRowOffsets 和 csrColInd 的基准索引
valueType	主机	输出	csrValues 的数据类型

有关返回状态的描述，请参见mcsparseStatus_t。

14.3.7 mcsparseCsrSetPointers()

```
mcsparseStatus_t
mcsparseCsrSetPointers(mcsparseSpMatDescr_t spMatDescr,
                       void*                csrRowOffsets,
                       void*                csrColInd,
                       void*                csrValues)
```

此函数设置稀疏矩阵描述符 spMatDescr 的指针。

参数	内存	输入/输出	含义
spMatDescr	主机	输入	稀疏矩阵描述符
csrRowOffsets	设备	输入	稀疏矩阵的行偏移。大小为 rows + 1 的数组
csrColInd	设备	输入	稀疏矩阵的列索引。大小为 nnz 的数组
csrValues	设备	输入	稀疏矩阵的非零元素大小为 nnz 的数组

有关返回状态的描述，请参见mcsparseStatus_t。

14.3.8 mcsparseCscSetPointers()

```
mcsparseStatus_t
mcsparseCscSetPointers(mcsparseSpMatDescr_t spMatDescr,
                       void*                cscColOffsets,
                       void*                cscRowInd,
                       void*                cscValues)
```

此函数设置稀疏矩阵描述符 spMatDescr 的指针。

参数	内存	输入/输出	含义
spMatDescr	主机	输入	稀疏矩阵描述符
cscColOffsets	设备	输入	稀疏矩阵的列偏移。大小为 cols + 1 的数组
cscRowInd	设备	输入	稀疏矩阵的行索引。大小为 nnz 的数组
cscValues	设备	输入	稀疏矩阵的非零元素大小为 nnz 的数组

有关返回状态的描述，请参见mcsparseStatus_t。

14.3.9 mcsparseCooSetPointers()

```
mcsparseStatus_t
mcsparseCooSetPointers(mcsparseSpMatDescr_t spMatDescr,
                       void*                cooRows,
                       void*                cooColumns,
                       void*                cooValues)
```

此函数设置稀疏矩阵描述符 spMatDescr 的指针。

参数	内存	输入/输出	描述
spMatDescr	主机	输入	稀疏矩阵描述符
cooRows	设备	输入	稀疏矩阵的行索引。大小为 nnz 的数组
cooColumns	设备	输入	稀疏矩阵的列索引。大小为 nnz 的数组
cooValues	设备	输入	稀疏矩阵的值。大小为 nnz 的数组

有关返回状态的描述，请参见mcsparseStatus_t。

14.3.10 mcsparseSpMatGetSize()

```
mcsparseStatus_t
mcsparseSpMatGetSize(mcsparseSpMatDescr_t spMatDescr,
                     int64_t*             rows,
                     int64_t*             cols,
                     int64_t*             nnz)
```

此函数返回稀疏矩阵 spMatDescr 的大小。

参数	内存	输入/输出	描述
spMatDescr	主机	输入	稀疏矩阵描述符
rows	主机	输出	稀疏矩阵的行数
cols	主机	输出	稀疏矩阵的列数
nnz	主机	输出	稀疏矩阵的非零元素的个数

有关返回状态的描述，请参见mcsparseStatus_t。

14.3.11 mcsparseSpMatGetFormat()

```
mcsparseStatus_t
mcsparseSpMatGetFormat(mcsparseSpMatDescr_t spMatDescr,
                       mcsparseFormat_t*    format)
```

此函数返回稀疏矩阵描述符 spMatDescr 的 format 字段。

参数	内存	输入/输出	含义
spMatDescr	主机	输入	稀疏矩阵描述符
format	主机	输出	稀疏矩阵的存储格式

有关返回状态的描述，请参见mcsparseStatus_t。

14.3.12 mcsparseSpMatGetIndexBase()

```
mcsparseStatus_t
mcsparseSpMatGetIndexBase(mcsparseSpMatDescr_t spMatDescr,
                          mcsparseIndexBase_t* idxBase)
```

此函数返回稀疏矩阵描述符 spMatDescr 的 idxBase 字段。

参数	内存	输入/输出	含义
spMatDescr	主机	输入	稀疏矩阵描述符
idxBase	主机	输出	稀疏矩阵的基准索引

有关返回状态的描述，请参见mcsparseStatus_t。

14.3.13 mcsparseSpMatGetValues()

```
mcsparseStatus_t
mcsparseSpMatGetValues(mcsparseSpMatDescr_t spMatDescr,
                       void**               values)
```

此函数返回稀疏矩阵描述符 spMatDescr 的 values 字段。

参数	内存	输入/输出	含义
spMatDescr	主机	输入	稀疏矩阵描述符
values	设备	输出	稀疏矩阵的值。大小为 nnz 的数组

有关返回状态的描述，请参见mcsparseStatus_t。

14.3.14 mcsparseSpMatSetValues()

```
mcsparseStatus_t
mcsparseSpMatSetValues(mcsparseSpMatDescr_t spMatDescr,
                       void*                values)
```

此函数设置稀疏矩阵描述符 spMatDescr 的 values 字段。

参数	内存	输入/输出	含义
spMatDescr	主机	输入	稀疏矩阵描述符
values	设备	输入	稀疏矩阵的值。大小为 nnz 的数组

有关返回状态的描述，请参见mcsparseStatus_t。

14.4 稠密向量 APIs

本节将描述稠密向量描述符的 mcSPARSE 帮助函数。

14.4.1 mcsparseCreateDnVec()

```
mcsparseStatus_t
mcsparseCreateDnVec(mcsparseDnVecDescr_t* dnVecDescr,
                    int64_t               size,
                    void*                 values,
                    macaDataType          valueType)
```

此函数初始化稠密向量描述符 dnVecDescr。

参数	内存	输入/输出	含义
dnVecDescr	主机	输出	稠密向量描述符
size	主机	输入	稠密向量的大小
values	设备	输入	稠密向量的值。大小为 nnz 的数组
valueType	主机	输入	枚举符，指定 values 数据类型

注解： 如果描述符不会被用作例程的输出参数 (例如转换函数)，则可以安全地使用 const_cast 来移除输入指针的 const 限定。

有关返回状态的描述，请参见mcsparseStatus_t。

14.4.2 mcsparseDestroyDnVec()

```
mcsparseStatus_t
mcsparseDestroyDnVec(mcsparseDnVecDescr_t dnVecDescr)
```

此函数释放了为稠密向量描述符 dnVecDescr 分配的主机内存。

参数	内存	输入/输出	含义
dnVecDescr	主机	输入	稠密向量描述符

有关返回状态的描述，请参见mcsparseStatus_t。

14.4.3 mcsparseDnVecGet()

```
mcsparseStatus_t
mcsparseDnVecGet(mcsparseDnVecDescr_t dnVecDescr,
                 int64_t*             size,
                 void**               values,
                 macaDataType*        valueType)
```

此函数返回稠密向量描述符 dnVecDescr 的字段。

参数	内存	输入/输出	含义
dnVecDescr	主机	输入	稠密向量描述符
size	主机	输出	稠密向量的大小
values	设备	输出	稠密向量的值。大小为 nnz 的数组
valueType	主机	输出	枚举符，指定 values 数据类型

有关返回状态的描述，请参见mcsparseStatus_t。

14.4.4 mcsparseDnVecGetValues()

```
mcsparseStatus_t
mcsparseDnVecGetValues(mcsparseDnVecDescr_t dnVecDescr,
                       void**               values)
```

此函数返回稠密向量描述符 dnVecDescr 的 values 字段。

参数	内存	输入/输出	描述
dnVecDescr	主机	输入	稠密向量描述符
values	设备	输出	稠密向量的值

有关返回状态的描述，请参见mcsparseStatus_t。

14.4.5 mcsparseDnVecSetValues()

```
mcsparseStatus_t
mcsparseDnVecSetValues(mcsparseDnVecDescr_t dnVecDescr,
                       void*                values)
```

此函数设置了稠密向量描述符 dnVecDescr 的 values 字段。

参数	内存	输入/输出	含义
dnVecDescr	主机	输入	稠密向量描述符
values	设备	输入	稠密向量的值。大小为 size 的数组

有关返回状态的描述，请参见mcsparseStatus_t。

14.5 稠密矩阵 APIs

这个部分描述了用于稠密矩阵描述符的 mcSPARSE 辅助函数。

14.5.1 mcsparseCreateDnMat()

```
mcsparseStatus_t
mcsparseCreateDnMat(mcsparseDnMatDescr_t* dnMatDescr,
                    int64_t               rows,
                    int64_t               cols,
                    int64_t               ld,
                    void*                 values,
                    macaDataType          valueType,
                    mcsparseOrder_t       order)
```

此函数初始化了稠密矩阵描述符 dnMatDescr。

参数	内存	输入/输出	含义
dnMatDescr	主机	输出	稠密矩阵描述符
rows	主机	输入	稠密矩阵的行数
cols	主机	输入	稠密矩阵的列数
ld	主机	输入	稠密矩阵的主维度
values	设备	输入	稠密矩阵的值。大小为 size 的数组
valueType	主机	输入	指定 values 数据类型的枚举类型
order	主机	输入	指定稠密矩阵的内存布局的枚举类型

注解： 如果描述符不会用作例行程序 (例如转换函数) 的输出参数，则可以安全地将输入指针的 constness(const_cast) 去除。

有关返回状态的描述，请参见mcsparseStatus_t。

14.5.2 mcsparseDestroyDnMat()

```
mcsparseStatus_t
mcsparseDestroyDnMat(mcsparseDnMatDescr_t dnMatDescr)
```

此函数释放了为稠密矩阵描述符 dnMatDescr 分配的主机内存。

参数	内存	输入/输出	含义
dnMatDescr	主机	输入	稠密矩阵描述符

有关返回状态的描述，请参见mcsparseStatus_t。

14.5.3 mcsparseDnMatGet()

```
mcsparseStatus_t
mcsparseDnMatGet(mcsparseDnMatDescr_t dnMatDescr,
                 int64_t*             rows,
                 int64_t*             cols,
                 int64_t*             ld,
                 void**               values,
                 macaDataType*        type,
                 mcsparseOrder_t*     order)
```

此函数返回了稠密矩阵描述符 dnMatDescr 的字段。

参数	内存	输入/输出	含义
dnMatDescr	主机	输入	稠密矩阵描述符
rows	主机	输出	稠密矩阵的行数
cols	主机	输出	稠密矩阵的列数
ld	主机	输出	稠密矩阵的主维度
values	设备	输出	稠密矩阵的值。大小为 ld * cols 的数组
valueType	主机	输出	指定 values 数据类型的枚举类型
order	主机	输出	指定稠密矩阵的内存布局的枚举类型

有关返回状态的描述，请参见mcsparseStatus_t。

14.5.4 mcsparseDnMatGetValues()

```
mcsparseStatus_t
mcsparseDnMatGetValues(mcsparseDnMatDescr_t dnMatDescr,
                       void**               values)
```

此函数返回了稠密矩阵描述符 dnMatDescr 的字段。

参数	内存	输入/输出	含义
dnMatDescr	主机	输入	稠密矩阵描述符
values	设备	输出	稠密矩阵的值。大小为 ld * cols 的数组

有关返回状态的描述，请参见mcsparseStatus_t。

14.5.5 mcsparseDnSetValues()

```
mcsparseStatus_t
mcsparseDnMatSetValues(mcsparseDnMatDescr_t dnMatDescr,
                       void*                values)
```

此函数设置了稠密矩阵描述符 dnMatDescr 的 values 字段。

参数	内存	输入/输出	含义
dnMatDescr	主机	输入	稠密矩阵描述符
values	设备	输入	稠密矩阵的值。大小为 ld * cols 的数组

有关返回状态的描述，请参见mcsparseStatus_t。

14.5.6 mcsparseDnMatGetStridedBatch()

```
mcsparseStatus_t
mcsparseDnMatGetStridedBatch(mcsparseDnMatDescr_t dnMatDescr,
                             int*                 batchCount,
                             int64_t*             batchStride)
```

此函数返回了稠密矩阵描述符 dnMatDescr 的批次数和批次步长。

参数	内存	输入/输出	含义
dnMatDescr	主机	输入	稠密矩阵描述符
batchCount	主机	输出	稠密矩阵的批次数
batchStride	主机	输出	一个矩阵与批次中下一个矩阵之间的地址偏移量

有关返回状态的描述，请参见mcsparseStatus_t。

14.5.7 mcsparseDnMatSetStridedBatch()

```
mcsparseStatus_t
mcsparseDnMatSetStridedBatch(mcsparseDnMatDescr_t dnMatDescr,
                             int                  batchCount,
                             int64_t              batchStride)
```

此函数设置了稠密矩阵描述符 dnMatDescr 的批次数和批次步长。

参数	内存	输入/输出	含义
dnMatDescr	主机	输入	稠密矩阵描述符
batchCount	主机	输入	稠密矩阵的批次数
batchStride	主机	输入	一个矩阵与批次中下一个矩阵之间的的地址偏移量。 • 如果矩阵使用列优先布局，则 batchStride >= ld * cols • 如果不是，则 batchStride >= ld * rows

有关返回状态的描述，请参见mcsparseStatus_t。

14.6 通用 API 函数

14.6.1 mcsparseSparseToDense()

```
mcsparseStatus_t
mcsparseSparseToDense_bufferSize(mcsparseHandle_t           handle,
                                 mcsparseSpMatDescr_t       matA,
                                 mcsparseDnMatDescr_t       matB,
                                 mcsparseSparseToDenseAlg_t alg,
                                 size_t*                    bufferSize)
```

```
mcsparseStatus_t
mcsparseSparseToDense(mcsparseHandle_t           handle,
                      mcsparseSpMatDescr_t       matA,
                      mcsparseDnMatDescr_t       matB,
                      mcsparseSparseToDenseAlg_t alg,
                      void*                      buffer)
```

此函数将稀疏矩阵 matA 从 CSR、CSC 或 COO 格式转换为其密集表示 matB。目前不支持 Blocked-ELL 格式。

函数 mcsparseSparseToDense_bufferSize() 返回 mcsparseSparseToDense() 函数所需的工作空间大小。

参数	内存	输入/输出	含义
handle	主机	输入	处理 mcSPARSE 库上下文的句柄
matA	主机	输入	稀疏矩阵 A
matB	主机	输出	稠密矩阵 B
alg	主机	输入	计算的算法
bufferSize	主机	输出	mcsparseSparseTo Dense() 函数所需的工作空间字节数。
buffer	设备	输入	指向工作空间缓冲区的指针

mcsparseSparseToDense() 支持以下索引类型来表示稀疏矩阵 matA：

- 32 位索引 (MCSPARSE_INDEX_32I)
- 64 位索引 (MCSPARSE_INDEX_64I)

mcsparseSparseToDense() 支持以下数据类型：

A/B
MACA_R_16F
MACA_R_16BF
MACA_R_32F
MACA_R_64F
MACA_C_16F
MACA_C_16BF
MACA_C_32F
MACA_C_64F

mcsparseSparse2Dense() 支持以下算法：

算法	说明
MCSPARSE_SPARSETODENSE_ALG_DEFAULT	默认算法

mcsparseSparseToDense() 具有以下特性：

- 这个例程不需要额外的存储空间
- 这个例程支持异步执行
- 对于每次运行都提供确定 (位级相同) 的结果

mcsparseSparseToDense() 支持以下优化：

- 硬件内存压缩

有关返回状态的描述，请参见mcsparseStatus_t。

14.6.2 mcsparseDenseToSparse()

```
mcsparseStatus_t
mcsparseDenseToSparse_bufferSize(mcsparseHandle_t           handle,
                                 mcsparseDnMatDescr_t       matA,
                                 mcsparseSpMatDescr_t       matB,
                                 mcsparseDenseToSparseAlg_t alg,
                                 size_t*                    bufferSize)
```

```
mcsparseStatus_t
mcsparseDenseToSparse_analysis(mcsparseHandle_t           handle,
                               mcsparseDnMatDescr_t       matA,
                               mcsparseSpMatDescr_t       matB,
                               mcsparseDenseToSparseAlg_t alg,
                               void*                      buffer)
```

```
mcsparseStatus_t
mcsparseDenseToSparse_convert(mcsparseHandle_t           handle,
                              mcsparseDnMatDescr_t       matA,
                              mcsparseSpMatDescr_t       matB,
                              mcsparseDenseToSparseAlg_t alg,
                              void*                      buffer)
```

此函数将稠密矩阵 matA 转换为稀疏矩阵 matB，可以选择转换为 CSR、CSC、COO 或 Blocked-ELL 格式。

函数 mcsparseDenseToSparse_bufferSize() 返回 mcsparseDenseToSparse_bufferSize() 所需的工作空间大小。

函数 mcsparseDenseToSparse_analysis() 更新稀疏矩阵描述符 matB 中的非零元素数量。用户负责分配稀疏矩阵所需的内存：

- 分别为 CSC 和 CSR 的行/列索引以及值数组。
- COO 格式的行、列和值数组。
- Blocked-ELL 格式的列索引 (ellColInd) 和值数组 (ellValue)。

最后，我们调用 mcsparseDenseToSparse_convert() 来填充在前一步骤中分配的数组。

参数	内存	输入/输出	含义
handle	主机	输入	处理 mcSPARSE 库上下文的句柄
matA	主机	输入	稠密矩阵 A
matB	主机	输出	稀疏矩阵 B
alg	主机	输入	计算的算法
bufferSize	主机	输出	mcsparseDens eToSparse_analysis() 函数所需的工作空间字节数
buffer	设备	输入	指向工作空间缓冲区的指针

mcsparseDenseToSparse() 支持以下索引类型来表示稀疏向量 matB：

- 32 位索引 (MCSPARSE_INDEX_32I)
- 64 位索引 (MCSPARSE_INDEX_64I)

mcsparseDenseToSparse() 支持以下数据类型：

A/B
MACA_R_16F
MACA_R_16BF
MACA_R_32F
MACA_R_64F
MACA_C_16F
MACA_C_16BF
MACA_C_32F
MACA_C_64F

mcsparseDense2Sparse() 支持以下算法：

算法	说明
MCSPARSE_DENSETOSPARSE_ALG_DEFAULT	默认算法

mcsparseDenseToSparse() 具有以下特性：

- 这个例程不需要额外的存储空间
- 这个例程支持异步执行
- 对于每次运行都提供确定 (位级相同) 的结果

mcsparseDenseToSparse() 支持以下优化：

- 硬件内存压缩

有关返回状态的描述，请参见mcsparseStatus_t。

14.6.3 mcsparseAxpby()

```
mcsparseStatus_t
mcsparseAxpby(mcsparseHandle_t     handle,
              const void*          alpha,
              mcsparseSpVecDescr_t vecX,
              const void*          beta,
              mcsparseDnVecDescr_t vecY)
```

此函数计算稀疏向量 vecX 和稠密向量 vecY 的和

$$Y=\alpha X+\beta Y$$

即，

```
for i=0 to n-1
   Y[i] = beta * Y[i] for i=0 to nnz-1 Y[X_indices[i]] += alpha * X_
↪values[i]
```

参数	内存	输入/输出	含义
handle	主机	输入	处理 mcSPARSE 库上下文的句柄
alpha	主机或设备	输入	用于乘法计算类型的标量
vecX	主机	输入	稀疏矩阵 X
beta	主机或设备	输入	用于乘法计算类型的标量
vecY	主机	输入/输出	稠密向量 Y

mcsparseAxpby 支持以下索引类型来表示稀疏向量 vecX：

- 32 位索引 (MCSPARSE_INDEX_32I)
- 64 位索引 (MCSPARSE_INDEX_64I)

mcsparseAxpby 支持以下数据类型：

统一精度计算：

X/Y/compute
MACA_R_32F
MACA_R_64F
MACA_C_32F
MACA_C_64F

混合精度计算：

X/ Y	compute
MACA_R_16F	MACA_R_32F
MACA_R_16BF	
MACA_C_16F	MACA_C_32F
MACA_C_16BF	

mcsparseAxpby() 有以下的约束条件：

- 表示稀疏向量 vecX 的数组必须对齐到 16 字节

mcsparseAxpby() 有以下特性：

- 这个例程不需要额外的存储空间
- 这个例程支持异步执行
- 如果稀疏向量 vecX 的索引是显著不同的，对于每次运行都提供确定 (位级相同) 的结果

mcsparseAxpby() 支持以下优化：

- 硬件内存压缩

有关返回状态的描述，请参见mcsparseStatus_t。

14.6.4 mcsparseGather()

```
mcsparseStatus_t
mcsparseGather(mcsparseHandle_t     handle,
               mcsparseDnVecDescr_t vecY,
               mcsparseSpVecDescr_t vecX)
```

此函数将稠密向量 vecY 的元素存储到稀疏向量 vecX 中。

即，

```
for i=0 to nnz-1
  X_values[i] = Y[X_indices[i]]
```

参数	内存	输入/输出	含义
handle	主机	输入	mcSPARSE 库上下文的句柄
vecX	主机	输出	稀疏向量 X
vecY	主机	输入	稠密向量 Y

mcsparseGather 支持以下索引类型来表示稀疏向量 vecX：

- 32 位索引 (MCSPARSE_INDEX_32I)
- 64 位索引 (MCSPARSE_INDEX_64I)

mcsparseGather 支持以下数据类型：

X/ Y
MACA_R_16F
MACA_R_16BF
MACA_R_32F
MACA_R_64F
MACA_C_16F
MACA_C_16BF
MACA_C_32F
MACA_C_64F

mcsparseGather() 有以下约束条件：

- 表示稀疏向量 vecX 的数组必须按照 16 字节对齐

mcsparseGather() 具有以下特性：

- 此函数不需要额外的存储空间
- 此函数支持异步执行
- 如果稀疏向量 vecX 的索引是显著不同的，每次运行都会得到确定性 (逐位) 的结果

mcsparseGather() 支持以下优化：

- 硬件内存压缩 (Hardware Memory Compression)

有关返回状态的描述，请参见mcsparseStatus_t。

14.6.5 mcsparseScatter()

```
mcsparseStatus_t
mcsparseScatter(mcsparseHandle_t     handle,
                mcsparseSpVecDescr_t vecX,
                mcsparseDnVecDescr_t vecY)
```

函数将稀疏向量 vecX 的元素散布到稠密向量 vecY 中。

即，

```
for i=0 to nnz-1
  Y[X_indices[i]] = X_values[i]
```

参数	内存	输入/输出	含义
handle	主机	输入	mcSPARSE 库上下文的句柄
vecX	主机	输入	稀疏向量 X
vecY	主机	输出	稠密向量 Y

mcsparseScatter 支持以下索引类型来表示稀疏向量 vecX：

- 32 位索引 (MCSPARSE_INDEX_32I)
- 64 位索引 (MCSPARSE_INDEX_64I)

mcsparseScatter 支持以下数据类型：

X/Y
MACA_R_16F
MACA_R_16BF
MACA_R_32F
MACA_R_64F
MACA_C_16F
MACA_C_16BF
MACA_C_32F
MACA_C_64F

mcsparseScatter() 有以下约束条件：

- 表示稀疏向量 vecX 的数组必须按照 16 字节对齐

mcsparseScatter() 具有以下特性：

- 此函数不需要额外的存储空间
- 此函数支持异步执行
- 如果稀疏向量 vecX 的索引是显著不同的，每次运行都会得到确定性 (逐位) 的结果

mcsparseScatter() 支持以下优化：

- 硬件内存压缩 (Hardware Memory Compression)

有关返回状态的描述，请参见mcsparseStatus_t。

14.6.6 mcsparseRot()

```
mcsparseStatus_t
mcsparseRot(mcsparseHandle_t     handle,
            const void*          c_coeff,
            const void*          s_coeff,
            mcsparseSpVecDescr_t vecX,
            mcsparseDnVecDescr_t vecY)
```

此函数用于计算 Givens 旋转矩阵。

即，

```
for i=0 to nnz-1
   Y[X_indices[i]] = c * Y[X_indices[i]] - s * X_values[i] = c * X_
↪values[i] + s * Y[X_indices[i]]
```

参数	内存	输入/输出	含义
handle	主机	输入	处理 mcSPARSE 库上下文的句柄
c_coeff	主机或设备	输入	旋转矩阵的余弦元素
vecX	主机	输入/输出	稀疏向量 X
s_coeff	主机或设备	输入	旋转矩阵的正弦元素
vecY	主机	输入/输出	稠密向量 Y

mcsparseRot 支持以下索引类型来表示稀疏向量 vecX：

- 32 位索引 (MCSPARSE_INDEX_32I)
- 64 位索引 (MCSPARSE_INDEX_64I)

mcsparseRot 支持以下数据类型：

统一精度计算：

X/Y/compute
MACA_R_32F
MACA_R_64F
MACA_C_32F
MACA_C_64F

混合精度计算：

X/Y	compute
MACA_R_16F	MACA_R_32F
MACA_R_16BF	
MACA_C_16F	MACA_C_32F
MACA_C_16BF	

mcsparseRot() 有以下约束条件：

- 表示稀疏向量 vecX 的数组必须按照 16 字节对齐

mcsparseRot() 具有以下特性：

- 此函数不需要额外的存储空间
- 此函数支持异步执行
- 如果稀疏向量 vecX 的索引是显著不同的，每次运行都会得到确定性 (逐位) 的结果

mcsparseRot() 支持以下优化：

- 硬件内存压缩 (Hardware Memory Compression)

有关返回状态的描述，请参见mcsparseStatus_t。

14.6.7 mcsparseSpVV()

```
mcsparseStatus_t
mcsparseSpVV_bufferSize(mcsparseHandle_t     handle,
                        mcsparseOperation_t  opX,
                        mcsparseSpVecDescr_t vecX,
                        mcsparseDnVecDescr_t vecY,
                        void*                result,
                        macaDataType         computeType,
                        size_t*              bufferSize)
```

```
mcsparseStatus_t
mcsparseSpVV(mcsparseHandle_t     handle,
             mcsparseOperation_t  opX,
             mcsparseSpVecDescr_t vecX,
             mcsparseDnVecDescr_t vecY,
             void*                result,
             macaDataType         computeType,
             void*                externalBuffer)
```

此函数计算稀疏向量 vecX 和稠密向量 vecY 的内积。

```
result = 0;
for i=0 to nnz-1
      result += X_values[i] * Y[X_indices[i]]
```

mcsparseSpVV_bufferSize() 函数返回 mcsparseSpVV() 所需的工作空间大小。

参数	内存	输入/输出	含义
handle	主机	输入	处理 mcSPARSE 库上下文的句柄
opX	主机	输入	操作类型 op(X) 可选非转置或共轭转置
vecX	主机	输入	稀疏向量 X
vecY	主机	输入	稠密向量 Y
result	主机或设备	输出	计算结果的点积
computeType	主机	输入	计算数据类型执行计算的类型
bufferSize	主机	输出	mcsparseSpVV 所需工作空间的字节数
externalBuffer	设备	输入	外部缓冲区的指针，至少为 bufferSize 字节大小的工作空间缓冲区

mcsparseSpVV 支持以下索引类型来表示稀疏向量 vecX：

- 32 位索引 (MCSPARSE_INDEX_32I)
- 64 位索引 (MCSPARSE_INDEX_64I)

mcsparseSpVV 当前支持以下数据类型的组合：

统一精度计算：

X/Y/computeType
MACA_R_32F
MACA_R_64F
MACA_C_32F
MACA_C_64F

混合精度计算：

X/Y	computeType/ result
MACA_R_8I	MACA_R_32I
MACA_R_8I	MACA_R_32F
MACA_R_16F	
MACA_R_16BF	
MACA_C_16F	MACA_C_32F
MACA_C_16BF	

mcsparseSpVV() 有以下约束条件：

- 表示稀疏向量 vecX 的数组必须按照 16 字节对齐

mcsparseSpVV() 具有以下特性：

- 此函数不需要额外的存储空间
- 此函数支持异步执行
- 如果稀疏向量 vecX 的索引是显著不同的，每次运行都会得到确定性 (逐位) 的结果

mcsparseSpVV() 支持以下优化：

- 硬件内存压缩 (Hardware Memory Compression)

有关返回状态的描述，请参见mcsparseStatus_t。

14.6.8 mcsparseSpMV()

```
mcsparseStatus_t
mcsparseSpMV_bufferSize(mcsparseHandle_t     handle,
                        mcsparseOperation_t  opA,
                        const void*          alpha,
                        mcsparseSpMatDescr_t matA,
                        mcsparseDnVecDescr_t vecX,
                        const void*          beta,
                        mcsparseDnVecDescr_t vecY,
                        macaDataType         computeType,
                        mcsparseSpMVAlg_t    alg,
                        size_t*              bufferSize)
```

```
mcsparseStatus_t
mcsparseSpMV(mcsparseHandle_t     handle,
             mcsparseOperation_t  opA,
             const void*          alpha,
             mcsparseSpMatDescr_t matA,
             mcsparseDnVecDescr_t vecX,
             const void*          beta,
             mcsparseDnVecDescr_t vecY,
             macaDataType         computeType,
             mcsparseSpMVAlg_t    alg,
             void*                externalBuffer)
```

此函数用于计算稀疏矩阵 matA 与稠密向量 vecX 的乘积。

函数 mcsparseSpMV_bufferSize() 返回 mcsparseSpMV() 所需的工作空间大小。

参数	内存	输入/输出	含义
handle	主机	输入	处理 mcSPARSE 库上下文的句柄
opA	主机	输入	操作符 op(A)
alpha	主机或设备	输入	用于 computeType 类型乘法的标量
matA	主机	输入	稀疏矩阵 A
vecX	主机	输入	稠密向量 X
beta	主机或设备	输入	用于 computeType 类型乘法的标量
vecY	主机	输入/输出	稠密向量 Y
computeType	主机	输入	计算执行的数据类型
alg	主机	输入	用于计算的算法
bufferSize	主机	输出	mcsparseSpMV 所需的工作空间大小以字节为单位
externalBuffer	设备	输入	指向至少大小为 bufferSize 字节的工作空间缓冲区的指针

目前支持的稀疏矩阵格式如下：

- MCSPARSE_FORMAT_COO
- MCSPARSE_FORMAT_CSR

mcsparseSpMV 支持以下索引类型来表示稀疏矩阵 matA：

- 32 位索引 (MCSPARSE_INDEX_32I)
- 64 位索引 (MCSPARSE_INDEX_64I)

mcsparseSpMV 支持以下数据类型：

统一精度计算：

A/X/ Y/computeType
MACA_R_32F
MACA_R_64F
MACA_C_16F
MACA_C_16BF
MACA_C_32F
MACA_C_64F

混合精度计算：

A/ X	Y	computeType
MACA_R_8I	MACA_R_32I	MACA_R_32I
MACA_R_8I	MACA_R_32F	MACA_R_32F
MACA_R_16F		
MACA_R_16BF		
MACA_R_16F	MACA_R_16F	
MACA_R_16BF	MACA_R_16BF	

混合常规/复数计算：

A	X/ Y/ computeType
MACA_R_32F	MACA_C_32F
MACA_R_64F	MACA_C_64F

mcsparseSpMV() 支持以下算法：

Algorithm	Notes
MCSPARSE_MV_ALG_DEFAULT MCSPARSE_SPMV_ALG_DEFAULT	适用于任何稀疏矩阵格式的默认算法
MCSPARSE_COOMV_ALG MCSPARSE_SPMV_COO_ALG1	COO 稀疏矩阵格式的默认算法可能在相同输入参数下，不同运行产生稍微不同的结果
MCSPARSE_SPMV_COO_ALG2	每次运行提供确定性 (逐位) 结果如果 opA != MCSPARSE_ OPERATION_NON_TRANSPOSE，则与 MCSPARSE_SPMV_COO_ALG1 相同
MCSPARSE_CSRMV_ALG1 MCSPARSE_SPMV_CSR_ALG1	CSR 稀疏矩阵格式的默认算法可能在相同输入参数下，不同运行产生稍微不同的结果
MCSPARSE_CSRMV_ALG2 MCSPARSE_SPMV_CSR_ALG2	每次运行提供确定性 (逐位) 结果如果 opA != MCSPARSE_ OPERATION_NON_TRANSPOSE，则与 MCSPARSE_SPMV_COO_ALG1 相同

性能注意事项：

- MCSPARSE_SPMV_COO_ALG1 和 MCSPARSE_SPMV_CSR_ALG1 比 MCSPARSE_SPMV_COO_ALG2 和 MCSPARSE_SPMV_CSR_ALG2 具有更高的性能。
- 一般而言，opA == MCSPARSE_OPERATION_NON_TRANSPOSE 相对于 opA != MCSPARSE_OPERATION_NON_TRANSPOSE 具有 3 倍的速度优势。

mcsparseSpMV() 具有以下特性：

- 该例程在 CSR 格式 (所有算法) 和 COO 格式中需要额外的存储空间，其中 COO 格式使用 MCSPARSE_SPMM_COO_ALG2 算法。
- 仅对于 MCSPARSE_SPMM_COO_ALG2 和 MCSPARSE_SPMM_CSR_ALG2 算法，以及 opA == MCSPARSE_OPERATION_NON_TRANSPOSE 情况下，提供每次运行的确定性 (按位) 结果。
- 该例程支持异步执行。

mcsparseSpMV() 支持以下优化：

- 硬件内存压缩 (Hardware Memory Compression)

有关返回状态的描述，请参见mcsparseStatus_t。

14.6.9 mcsparseSpSV()

```
mcsparseStatus_t
mcsparseSpSV_createDescr(mcsparseSpSVDescr_t* spsvDescr);

mcsparseStatus_t
mcsparseSpSV_destroyDescr(mcsparseSpSVDescr_t spsvDescr);
```

```
mcsparseStatus_t
mcsparseSpSV_bufferSize(mcsparseHandle_t     handle,
                        mcsparseOperation_t  opA,
                        const void*          alpha,
                        mcsparseSpMatDescr_t matA,
                        mcsparseDnVecDescr_t vecX,
                        mcsparseDnVecDescr_t vecY,
                        macaDataType         computeType,
                        mcsparseSpSVAlg_t    alg,
                        mcsparseSpSVDescr_t  spsvDescr,
                        size_t*              bufferSize)
```

```
mcsparseStatus_t
mcsparseSpSV_analysis(mcsparseHandle_t     handle,
                      mcsparseOperation_t  opA,
                      const void*          alpha,
                      mcsparseSpMatDescr_t matA,
                      mcsparseDnVecDescr_t vecX,
                      mcsparseDnVecDescr_t vecY,
                      macaDataType         computeType,
                      mcsparseSpSVAlg_t    alg,
                      mcsparseSpSVDescr_t  spsvDescr
                      void*                externalBuffer)
```

```
mcsparseStatus_t
mcsparseSpSV_solve(mcsparseHandle_t     handle,
                   mcsparseOperation_t  opA,
                   const void*          alpha,
                   mcsparseSpMatDescr_t matA,
                   mcsparseDnVecDescr_t vecX,
                   mcsparseDnVecDescr_t vecY,
                   macaDataType         computeType,
                   mcsparseSpSVAlg_t    alg,
                   mcsparseSpSVDescr_t  spsvDescr)
```

此函数用于求解一个线性方程组，其系数用稀疏三角形矩阵表示。

函数 mcsparseSpSV_bufferSize() 返回 mcsparseSpSV_analysis() 和 mcsparseSpSV_solve() 所需的工作空间大小。函数 mcsparseSpSV_analysis() 执行分析阶段，而 mcsparseSpSV_solve() 执行稀疏三角形线性系统的求解阶段。不透明数据结构 spsvDescr 用于在所有函数之间共享信息。

该例程支持输入矩阵的任意稀疏性，但计算时只考虑上三角或下三角部分。

mcsparseSpSV API 调用中的所有参数必须一致，并且不得在 mcsparseSpSV_analysis() 和 mcsparseSpSV_solve() 之间修改矩阵描述。

参数	内存	输入/输出	含义
handle	主机	输入	处理 MCSPARSE 库上下文句柄
opA	主机	输入	操作 op(A)
alpha	主机或设备	输入	用于 computeType 类型乘法的标量
matA	主机	输入	稀疏矩阵 A
vecX	主机	输入	稠密矢量 X
vecY	主机	输入/输出	密集矢量 Y
computeType	主机	输入	执行计算的数据类型
alg	主机	输入	计算算法
bufferSize	主机	输出	mcsparseSpSV_ analysis() 和 mcsparseSpSV _solve() 所需的工作空间字节数
ex ternalBuffer	设备	输入	工作区缓冲区的指针，其大小至少为 bufferSize 字节。它被 mcsparse SpSV_analysis 和 mcsparse SpSV_solve() 使用。
spsvDescr	主机	输入/输出	不透明描述符，用于存储三个步骤中使用的内部数据

目前支持的稀疏矩阵格式如下：

- MCSPARSE_FORMAT_CSR
- MCSPARSE_FORMAT_COO

mcsparseSpSV() 支持以下类型和属性：

- MCSPARSE_FILL_MODE_LOWER 和 MCSPARSE_FILL_MODE_UPPER 填充模式
- MCSPARSE_DIAG_TYPE_NON_UNIT 和 MCSPARSE_DIAG_TYPE_UNIT 对角线类型

mcsparseSpSV() 支持以下索引类型来表示稀疏矩阵 matA：

- 32-bit indices (MCSPARSE_INDEX_32I)
- 64-bit indices (MCSPARSE_INDEX_64I)

mcsparseSpSV() 支持以下数据类型：

统一精度计算：

A/X/ Y/computeType
MACA_R_32F
MACA_R_64F
MACA_C_32F
MACA_C_64F

mcsparseSpSV() 支持以下算法：

算法注记	
MCSPARSE_SPSV_ALG_DEFAULT	默认算法

mcsparseSpSV() 具有以下属性：

- 该例程在分析阶段需要额外的存储空间，该存储空间大小与稀疏矩阵的非零条目数量成正比
- 为求解阶段 mcsparseSpSV_solve() 的每次运行提供确定性 (按位) 结果
- 该例程支持异步执行

mcsparseSpSV() 支持以下优化：

- 硬件内存压缩

有关返回状态的描述，请参见mcsparseStatus_t。

14.6.10 mcsparseSpMM()

```
mcsparseStatus_t
mcsparseSpMM_bufferSize(mcsparseHandle_t     handle,
                        mcsparseOperation_t  opA,
                        mcsparseOperation_t  opB,
                        const void*          alpha,
                        mcsparseSpMatDescr_t matA,
                        mcsparseDnMatDescr_t matB,
                        const void*          beta,
                        mcsparseDnMatDescr_t matC,
                        macaDataType         computeType,
                        mcsparseSpMMAlg_t    alg,
                        size_t*              bufferSize)
```

```
mcsparseStatus_t
mcsparseSpMM_preprocess(mcsparseHandle_t      handle,
                        mcsparseOperation_t   opA,
                        mcsparseOperation_t   opB,
                        const void*           alpha,
                        mcsparseSpMatDescr_t  matA,
                        mcsparseDnMatDescr_t  matB,
                        const void*           beta,
                        mcsparseDnMatDescr_t  matC,
                        macaDataType          computeType,
                        mcsparseSpMMAlg_t     alg,
                        void*                 externalBuffer)
```

```
mcsparseStatus_t
mcsparseSpMM(mcsparseHandle_t     handle,
             mcsparseOperation_t  opA,
             mcsparseOperation_t  opB,
             const void*          alpha,
             mcsparseSpMatDescr_t matA,
             mcsparseDnMatDescr_t matB,
             const void*          beta,
             mcsparseDnMatDescr_t matC,
             macaDataType         computeType,
             mcsparseSpMMAlg_t    alg,
             void*                externalBuffer)
```

此函数执行稀疏矩阵 matA 和稠密矩阵 matB 的乘法运算。

通过切换稠密矩阵的布局，该例程还可用于执行稠密矩阵 matB 和稀疏矩阵 matA 的乘法运算。

函数 mcsparseSpMM_bufferSize() 返回 mcsparseSpMM() 所需的工作空间大小。mcsparseSpMM_preprocess() 函数可以在 mcsparseSpMM 之前调用，以加快实际计算速度。当使用相同的稀疏性模式 (matA) 多次调用 mcsparseSpMM 时，它非常有用。稠密矩阵 (matB, matC) 的值可以任意变化。它需要 MCSPARSE_SPMM_CSR_ALG3。所有其他格式和算法均无效。

参数	内存	输入	含义
handle	主机	输入	处理 MCSPARSE 库上下文句柄
opA	主机	输入	操作 op(A)
opB	主机	输入	操作 op(A)
alpha	主机或设备	输入输入	用于 computeType 类型乘法的标量
matA	主机	输入	稀疏矩阵 A
matB	主机	输入	稠密矩阵 B
beta	主机或设备	输入	用于 computeType 类型乘法的标量
matC	主机	输入/输出	稠密矩阵 C
computeType	主机	输入	执行计算的数据类型
alg	主机	输入	计算算法
bufferSize	主机	输出	mcsparseSpMM 需要的工作空间字节数
externalBuffer	设备	输入	工作区缓冲区的指针，其大小至少为 bufferSize 字节。

14.6.11 mcsparseSpSM()

```
mcsparseStatus_t
mcsparseSpSM_createDescr(mcsparseSpSMDescr_t* spsmDescr);

mcsparseStatus_t
mcsparseSpSM_destroyDescr(mcsparseSpSMDescr_t spsmDescr);
```

```
mcsparseStatus_t
mcsparseSpSM_bufferSize(mcsparseHandle_t     handle,
                        mcsparseOperation_t  opA,
                        mcsparseOperation_t  opB,
                        const void*          alpha,
                        mcsparseSpMatDescr_t matA,
                        mcsparseDnMatDescr_t matB,
                        mcsparseDnMatDescr_t matC,
                        macaDataType         computeType,
                        mcsparseSpSMAlg_t    alg,
                        mcsparseSpSMDescr_t  spsmDescr,
                        size_t*              bufferSize)
```

```
mcsparseStatus_t
mcsparseSpSM_analysis(mcsparseHandle_t     handle,
                      mcsparseOperation_t  opA,
                      mcsparseOperation_t  opB,
                      const void*          alpha,
                      mcsparseSpMatDescr_t matA,
                      mcsparseDnMatDescr_t matB,
                      mcsparseDnMatDescr_t matC,
                      macaDataType         computeType,
                      mcsparseSpSMAlg_t    alg,
                      mcsparseSpSMDescr_t  spsmDescr,
                      void*                externalBuffer)
```

```
mcsparseStatus_t
mcsparseSpSM_solve(mcsparseHandle_t     handle,
                   mcsparseOperation_t  opA,
                   mcsparseOperation_t  opB,
```

(下页继续)

(续上页)

```
                    const void*           alpha,
                    mcsparseSpMatDescr_t  matA,
                    mcsparseDnMatDescr_t  matB,
                    mcsparseDnMatDescr_t  matC,
                    macaDataType          computeType,
                    mcsparseSpSMAlg_t     alg,
                    mcsparseSpSMDescr_t   spsmDescr)
```

此函数求解一个线性方程组，该方程组的系数用稀疏三角矩阵表示。

函数 mcsparseSpSM_bufferSize() 返回 mcsparseSpSM_analysis() 和 mcsparseSpSM_solve() 所需工作空间的大小。函数 mcsparseSpSM_analysis() 执行分析阶段，而 mcsparseSpSM_solve() 执行稀疏三角线性系统的求解阶段。不透明数据结构 spsmDescr 用于在所有函数之间共享信息。

该例程支持输入矩阵的任意稀疏性，但在计算中只考虑上或下三角部分。mcsparseSpSM_bufferSize() 要求分析阶段的缓冲区大小与稀疏矩阵的非零元素的数量成正比。

externalBuffer 被存储在 spsmDescr 中，并被 mcsparseSpSM_solve() 使用。因此，只有在 mcsparseSpSM_solve() 之后才能释放设备内存缓冲区。

注解: 所有参数必须在 mcsparseSpSM API 调用中保持一致，并且不得在 mcsparseSpSM_analysis() 和 mcsparseSpSM_solve() 之间之间不得修改矩阵描述。

参数	存储	输入/输出	含义
handle	主机	输入	处理 mcSPARSE 库上下文的句柄
opA	主机	输入	操作 op(A)
opB	主机	输入	操作 op(B)
alpha	主机或设备	输入	用于 computeType 类型乘法的标量
matA	主机	输入	稀疏矩阵 A
matB	主机	输入	稠密矩阵 B
matC	主机	输入/输出	稠密矩阵 C
computeType	主机	输入	执行计算的数据类型
alg	主机	输入	计算算法
bufferSize	主机	输出	mcsparseSp SM_analysis() 和 mcsparse SpSM_solve() 所需工作空间的字节数
externalBuffer	设备	输入	指向至少为 bufferSize 字节的工作区缓冲区的指针它由 mcsparse SpSM_analysis 和 mcsparse SpSM_solve() 使用
spsmDescr	主机	输入/输出	用于存储跨三个步骤使用的内部数据的不透明描述符

目前支持的稀疏矩阵格式如下：

- MCSPARSE_FORMAT_CSR
- MCSPARSE_FORMAT_COO

mcsparseSpSM() 支持以下形状和属性：

- MCSPARSE_FILL_MODE_LOWER 和 MCSPARSE_FILL_MODE_UPPER 填充模式
- MCSPARSE_DIAG_TYPE_NON_UNIT 和 MCSPARSE_DIAG_TYPE_UNIT 对角线类型

填充模式和对角线类型可以通过 mcsparseSpMatSetAttribute() 来设置

mcsparseSpSM() 支持以下索引类型表示稀疏矩阵 matA：

- 32 位索引 (MCSPARSE_INDEX_32I)
- 64 位索引 (MCSPARSE_INDEX_64I)

mcsparseSpSM() 支持以下数据类型：

统一精度计算：

A/B/ C/computeType
MACA_R_32F
MACA_R_64F
MACA_C_32F
MACA_C_64F

mcsparseSpSM() 支持以下算法：

算法注记	
MCSPARSE_SPSM_ALG_DEFAULT	默认算法

mcsparseSpSM() 具有以下特性：

- 该例程不需要额外的存储空间
- 对于求解阶段的每次运行，提供确定性 (按位) 结果的 mcsparseSpSM_solve() 函数
- 该例程支持异步执行
- 硬件内存压缩

有关返回状态的描述，请参见mcsparseStatus_t。

14.6.12 mcsparseSDDMM()

```
mcsparseStatus_t
mcsparseSDDMM_bufferSize(mcsparseHandle_t     handle,
                         mcsparseOperation_t  opA,
                         mcsparseOperation_t  opB,
                         const void*          alpha,
                         mcsparseDnMatDescr_t matA,
                         mcsparseDnMatDescr_t matB,
                         const void*          beta,
                         mcsparseSpMatDescr_t matC,
                         macaDataType         computeType,
                         mcsparseSDDMMAlg_t   alg,
                         size_t*              bufferSize)
```

```
mcsparseStatus_t
mcsparseSDDMM_preprocess(mcsparseHandle_t     handle,
                         mcsparseOperation_t  opA,
                         mcsparseOperation_t  opB,
                         const void*          alpha,
                         mcsparseDnMatDescr_t matA,
                         mcsparseDnMatDescr_t matB,
                         const void*          beta,
                         mcsparseSpMatDescr_t matC,
                         macaDataType         computeType,
```

(下页继续)

(续上页)

```
                            mcsparseSDDMMAlg_t   alg,
                            void*                externalBuffer)
```

```
mcsparseStatus_t
mcsparseSDDMM(mcsparseHandle_t     handle,
              mcsparseOperation_t  opA,
              mcsparseOperation_t  opB,
              const void*          alpha,
              mcsparseDnMatDescr_t matA,
              mcsparseDnMatDescr_t matB,
              const void*          beta,
              mcsparseSpMatDescr_t matC,
              macaDataType         computeType,
              mcsparseSDDMMAlg_t   alg,
              void*                externalBuffer)
```

此函数执行 matA 和 matB 的乘法运算，然后执行 matC 的稀疏模式的元素乘法。

函数 mcsparseSDDMM_bufferSize() 返回 mcsparseSDDMM 或 mcsparseSDDMM_preprocess 所需工作空间的大小。

mcsparseSDDMM_preprocess() 函数可以在 mcsparseSDDMM 之前被调用来加速实际计算。当使用相同的稀疏模式 (matC) 多次调用 mcsparseSDDMM 时，这将非常有用。稠密矩阵 (matA，matB) 的值可以任意更改。

参数	存储	输入/输出	含义
handle	主机	输入	处理 mcSPARSE 库上下文的句柄
opA	主机	输入	操作 op(A)
opB	主机	输入	操作 op(B)
alpha	主机 or 设备	输入	用于 computeType 类型乘法的标量。
matA	主机	输入	稠密矩阵 matA
matB	主机	输入	稠密矩阵 matB
beta	主机 or 设备	输入	用于 computeType 类型乘法的标量。
matC	主机	输入/输出	稀疏矩阵 matC
computeType	主机	输入	执行计算的数据类型。
alg	主机	输入	计算算法
bufferSize	主机	输出	mcsparseSDDMM 所需工作空间的字节数。
externalBuffer	设备	输入	指向至少为 bufferSize 字节的工作区缓冲区的指针

目前支持的稀疏矩阵格式：

- MCSPARSE_FORMAT_CSR

mcsparseSpSV() 支持以下索引类型来表示稀疏矩阵 matA：

- 32 位索引 (MCSPARSE_INDEX_32I)
- 64 位索引 (MCSPARSE_INDEX_64I)

mcsparseSDDMM 目前支持的数据类型组合如下：

统一精度计算：

A/X/ Y/computeType
MACA_R_32F
MACA_R_64F
MACA_C_32F
MACA_C_64F

mcsparseSpSV() 支持以下算法：

算法注记	
MCSPARSE_SDDMM_ALG_DEFAULT	默认算法

mcsparseSDDMM() 具有以下属性：

- 该例程不需要额外的存储空间
- 为每次运行提供确定的 (按位) 结果
- 例程支持异步执行

mcsparseSDDMM() 支持以下优化：

- 硬件内存压缩

有关返回状态的描述，请参见mcsparseStatus_t。

14.6.13 mcsparseSpGEMM()

```
mcsparseStatus_t
mcsparseSpGEMM_createDescr(mcsparseSpGEMMDescr_t* descr)

mcsparseStatus_t
mcsparseSpGEMM_destroyDescr(mcsparseSpGEMMDescr_t descr)
```

```
mcsparseStatus_t
mcsparseSpGEMM_workEstimation(mcsparseHandle_t      handle,
                              mcsparseOperation_t   opA,
                              mcsparseOperation_t   opB,
                              const void*           alpha,
                              mcsparseSpMatDescr_t  matA,
                              mcsparseSpMatDescr_t  matB,
                              const void*           beta,
                              mcsparseSpMatDescr_t  matC,
                              macaDataType          computeType,
                              mcsparseSpGEMMAlg_t   alg,
                              mcsparseSpGEMMDescr_t spgemmDescr,
                              size_t*               bufferSize1,
                              void*                 externalBuffer1)

mcsparseStatus_t
mcsparseSpGEMM_compute(mcsparseHandle_t      handle,
                       mcsparseOperation_t   opA,
                       mcsparseOperation_t   opB,
                       const void*           alpha,
                       mcsparseSpMatDescr_t  matA,
                       mcsparseSpMatDescr_t  matB,
                       const void*           beta,
                       mcsparseSpMatDescr_t  matC,
                       macaDataType          computeType,
                       mcsparseSpGEMMAlg_t   alg,
                       mcsparseSpGEMMDescr_t spgemmDescr,
                       void*                 externalBuffer1,
                       size_t*               bufferSize2,
                       void*                 externalBuffer2)
```

(下页继续)

(续上页)

```
mcsparseStatus_t
mcsparseSpGEMM_copy(mcsparseHandle_t      handle,
                    mcsparseOperation_t   opA,
                    mcsparseOperation_t   opB,
                    const void*           alpha,
                    mcsparseSpMatDescr_t  matA,
                    mcsparseSpMatDescr_t  matB,
                    const void*           beta,
                    mcsparseSpMatDescr_t  matC,
                    macaDataType          computeType,
                    mcsparseSpGEMMAlg_t   alg,
                    mcsparseSpGEMMDescr_t spgemmDescr)
```

此函数执行两个稀疏矩阵 matA 和 matB 的乘法运算。

函数 mcsparseSpGEMM_workEstimation() 和 mcsparseSpGEMM_compute() 被用于确定缓冲区大小和执行实际计算。

参数	内存	输入/输出	含义
handle	主机	输入	处理 mcSPARSE 库上下文句柄
opA	主机	输入	操作 op(A)
opB	主机	输入	操作 op(B)
alpha	主机或设备	输入	用于乘法的标量
matA	主机	输入	稀疏矩阵 A
matB	主机	输入	稀疏矩阵 B
beta	主机或设备	输入	用于乘法的标量
matC	主机	输入/输出	稀疏矩阵 C
computeType	主机	输入	枚举器，指定计算执行时使用的数据类型
alg	主机	输入	枚举器，指定计算所用算法。
spgemmDescr	主机	输入/输出	不透明描述符，用于存储在三个步骤之间使用的内部数据。
bufferSize1	主机	输入/输出	由 mcsparseSpGEMM_workEstimation 请求的工作空间字节数。
bufferSize2	主机	输入/输出	由 mcsparseS pGEMM_compute 请求的工作空间字节数。
externalBuffer1	设备	输入	mcsparseSpGEMM_workEstimation 和 mcsparseSpGEMM_compute 需要的工作空间缓冲区指针。
externalBuffer2	设备	输入	mcsparseSpGEMM_compute 和 mcsparseSpGEMM_copy 需要的工作空间缓冲区指针。

内存需求：第一次调用 mcsparseSpGEMM_compute 时会提供计算所需内存的 上限，通常是实际使用内存的几倍大。用户可以在第二次调用时提供任意大小的缓冲区 bufferSize2。如果内存大小不足，该例程将返回 MCSPARSE_STATUS_INSUFFICIENT_RESOURCES 状态。

目前，此函数具有以下限制：

- 只支持 32 位索引 MCSPARSE_INDEX_32I
- 只支持 CSR 格式 MCSPARSE_FORMAT_CSR
- 只支持 opA 和 opB 等于 MCSPARSE_OPERATION_NON_TRANSPOSE 的情况。

目前支持 mcsparseSpGEMM 函数的数据类型组合如下：

均匀精度计算：

A/B/ C/computeType
MACA_R_16F
MACA_R_16BF
MACA_R_32F
MACA_R_64F
MACA_C_16F
MACA_C_16BF
MACA_C_32F
MACA_C_64F

mcsparseSpGEMM 例程运行以下算法：

算法	注释
MCSPARSE_SPGEMM_DEFAULT	默认算法。为每次运行提供确定性 (位级别) 的结果。

mcsparseSpGEMM() 具有以下特性：

- 这个例程不需要额外的存储空间。
- 这个例程支持异步执行。

mcsparseSpGEMM() 支持以下优化：

- 硬件内存压缩。

有关返回状态的描述，请参见mcsparseStatus_t。

14.6.14 mcsparseSpGEMMreuse()

```
mcsparseStatus_t
mcsparseSpGEMM_createDescr(mcsparseSpGEMMDescr_t* descr)

mcsparseStatus_t
mcsparseSpGEMM_destroyDescr(mcsparseSpGEMMDescr_t descr)
```

```
mcsparseStatus_t
mcsparseSpGEMMreuse_workEstimation(mcsparseHandle_t      handle,
                                   mcsparseOperation_t   opA,
                                   mcsparseOperation_t   opB,
                                   mcsparseSpMatDescr_t  matA,
                                   mcsparseSpMatDescr_t  matB,
                                   mcsparseSpMatDescr_t  matC,
                                   mcsparseSpGEMMAlg_t   alg,
                                   mcsparseSpGEMMDescr_t spgemmDescr,
                                   size_t*               bufferSize1,
                                   void*                 externalBuffer1)

mcsparseStatus_t
mcsparseSpGEMMreuse_nnz(mcsparseHandle_t      handle,
                        mcsparseOperation_t   opA,
                        mcsparseOperation_t   opB,
                        mcsparseSpMatDescr_t  matA,
                        mcsparseSpMatDescr_t  matB,
                        mcsparseSpMatDescr_t  matC,
                        mcsparseSpGEMMAlg_t   alg,
                        mcsparseSpGEMMDescr_t spgemmDescr,
```

(下页继续)

(续上页)

```
                          size_t*              bufferSize2,
                          void*                externalBuffer2,
                          size_t*              bufferSize3,
                          void*                externalBuffer3,
                          size_t*              bufferSize4,
                          void*                externalBuffer4)

mcsparseStatus_t MCSPARSEAPI
mcsparseSpGEMMreuse_copy(mcsparseHandle_t      handle,
                         mcsparseOperation_t   opA,
                         mcsparseOperation_t   opB,
                         mcsparseSpMatDescr_t  matA,
                         mcsparseSpMatDescr_t  matB,
                         mcsparseSpMatDescr_t  matC,
                         mcsparseSpGEMMAlg_t   alg,
                         mcsparseSpGEMMDescr_t spgemmDescr,
                         size_t*               bufferSize5,
                         void*                 externalBuffer5)

mcsparseStatus_t MCSPARSEAPI
mcsparseSpGEMMreuse_compute(mcsparseHandle_t      handle,
                            mcsparseOperation_t   opA,
                            mcsparseOperation_t   opB,
                            const void*           alpha,
                            mcsparseSpMatDescr_t  matA,
                            mcsparseSpMatDescr_t  matB,
                            const void*           beta,
                            mcsparseSpMatDescr_t  matC,
                            macaDataType          computeType,
                            mcsparseSpGEMMAlg_t   alg,
                            mcsparseSpGEMMDescr_t spgemmDescr)
```

此函数执行两个稀疏矩阵 matA 和 matB 的乘法运算，其中输出矩阵 matC 的结构可重复用于不同值的多次计算。

函数 mcsparseSpGEMMreuse_workEstimation()，mcsparseSpGEMMreuse_nnz() 和 mcsparseSpGEMMreuse_copy() 用于确定缓冲区大小和执行实际计算。

参数	内存	输入/输出	描述
handle	主机	输入	处理 mcSPARSE 库上下文的句柄。
opA	主机	输入	操作 op(A)
opB	主机	输入	操作 op(B)
alpha	主机或设备	输入	用于乘法的标量
matA	主机	输入	稀疏矩阵 A
matB	主机	输入	稀疏矩阵 B
beta	主机或设备	输入	用于乘法的标量
matC	主机	输入/输出	稀疏矩阵 C
computeType	主机	输入	枚举器，指定计算执行的数据类型。
alg	主机	输入	枚举器，指定计算的算法。
spgemmDescr	主机	输入/输出	用于存储跨三个步骤使用的内部数据的不透明描述符。
bufferSize1	主机	输入/输出	mcsparseSpGEMMreuse_workEstimation 所需的工作空间字节数。
bufferSize2 bufferSize3 bufferSize4	主机	输入/输出	mcsparseSpGEMMreuse_nnz 所需的工作空间字节数。

下页继续

表 14.1 – 续上页

参数	内存	输入/输出	描述
bufferSize5	主机	输入/输出	mcsparseSpGEMMreuse_copy 所需的工作空间的字节数。
externalBuffer1	设备	输入	mcsparseSpGEMMreuse_workEstimation 和 mcsparseSpGEMMreuse_nnz 所需的工作空间的字节数。
externalBuffer2	设备	输入	mcsparseSpGEMMreuse_nnz 所需的工作空间的字节数。
externalBuffer3	设备	输入	mcsparseSpGEMMreuse_nnz 和 mcsparseSpGEMMreuse_copy 所需的工作空间的字节数。
externalBuffer4	设备	输入	mcsparseSpGEMMreuse_nnz 和 mcsparseSpGEMMreuse_compute 所需的工作空间的字节数。
externalBuffer5	设备	输入	指向 mcsparseSpGEMMreuse_copy 和 mcsparseSpGEMMreuse_compute 所需的工作空间缓冲区的指针。

内存需求：mcsparseSpGEMMreuse 需要将所有中间乘积保留在内存中，以便重复使用输出矩阵的结构。另一方面，一般情况下，中间乘积的数量比非零元素的数量高出几个数量级。为了最小化内存需求，该例程使用多个缓冲区，这些缓冲区可以在不再需要时可以释放。如果中间产品的数量超过 2^31-1，该例程将返回 MCSPARSE_STATUS_INSUFFICIENT_RESOURCES 状态。

目前，此函数有以下限制：

- 仅支持 32 位索引 MCSPARSE_INDEX_32I。
- 仅支持 CSR 格式 MCSPARSE_FORMAT_CSR。
- 仅支持 opA 和 opB 等于 MCSPARSE_OPERATION_NON_TRANSPOSE。

目前支持的 mcsparseSpGEMMreuse 的数据类型组合如下：

统一精度计算：

A/B/ C/computeType
MACA_R_32F
MACA_R_64F
MACA_C_16F
MACA_C_16BF
MACA_C_32F
MACA_C_64F

混合精度计算：

A/B	C	computeType
MACA_R_16F	MACA_R_16F	MACA_R_32F
MACA_R_16BF	MACA_R_16BF	MACA_R_32F

mcsparseSpGEMMreuse 例程运行以下算法：

Algorithm	注释
MCSPARSE_SPGEMM_DEFAULT MCSPARSE_SPGEMM_CSR_ALG_NONDETERMINITIC	默认算法。为每次运行提供确定性 (位级) 的输出矩阵结构，但值的算不是确定性的。
MCSPARSE_SPGEMM_CSR_ALG_DETERMINITIC	为每次运行提供确定性 (位级) 结构的输出矩阵和值计算。

`mcsparseSpGEMMreuse()` 具有以下特性：

- 该例程不需要额外的存储空间。
- 该例程支持异步执行。

`mcsparseSpGEMMreuse()` 支持以下优化：

- 硬件内存压缩。

有关返回状态的描述，请参见mcsparseStatus_t。