曦云® 系列通用计算 GPU

mcSOLVER API 参考

CSRD-23017-020-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	2024-01-10	描述性修改及格式修正 更新以下章节： • mcsolverDn<t>ormqr() • mcsolverDnXsytrs()
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

mcSOLVER 库是一个基于 mcBLAS 和 mcSPARSE 库的高级软件包。它在沐曦 MXMACA® Runtime（MXMACA Toolkit 的一部分）上实现了类似 LAPACK 的功能，例如稠密矩阵的常用矩阵分解和三角求解例程、稀疏最小二乘法求解器和特征值求解器等。包括两个模块，对应两个 API 集合：

1. 在单 GPU 上的 mcSOLVER API
2. 在单节点多 GPU 上的 mcsolverMG API

mcsolverMG 适用于多 GPU 应用程序。它在当前版本不受支持。

mcSOLVER 将三个独立的组件整合到了统一的框架下。mcSOLVER 的第一部分叫做 mcsolverDN，主要用于处理稠密矩阵分解和求解例程，例如 LU 分解、QR 分解、SVD 分解和 LDLT 分解，mcSOLVER 还提供了一些有用的工具函数，如矩阵和向量置换。其次，mcsolverSP 基于稀疏 QR 分解方法 (sparse QR factorization) 提供了一组新的稀疏例程 (sparse routines)。由于在因式分解中，并不是所有矩阵都具有适合并行处理 (parallelism) 的良好稀疏性模式 (sparsity pattern)，所以 mcsolverSP 库还提供了处理类顺序矩阵 (sequential-like matrices) 的 CPU 路径。对于那些具有高并行性的矩阵，这种 GPU 路径将提供更高的性能。最后一部分是 mcsolverRF，这是一个稀疏重因式分解（sparse re-factorization）软件包，在解决一系列具有相同稀疏性模式但系数变化的矩阵时可以提供非常好的性能。

当前版本暂不支持 mcsolverSp 和 mcsolverRf。

1.1 mcsolverDN：Dense LAPACK

mcsolverDN 库被设计用于求解形如以下的稠密线性系统 (dense linear systems)

$$Ax = b$$

其中系数矩阵 $A \in R^{nxn}$ ，右端向量 $b \in R^n$ 和结果向量 $x \in R^n$。mcsolverDN 库提供了 QR 分解 (QR factorization) 和使用部分主元消元法 (partial pivoting) 的 LU 分解以处理可能是非对称的一般矩阵 `A`。Cholesky 分解 (Cholesky factorization) 也用于对称/厄米特矩阵 (symmetric/Hermitian matrices)。对于对称不定矩阵 (symmetric indefinite matrices)，我们提供了 Bunch-Kaufman（LDL）分解。

mcsolverDN 库还提供了非常有用的双对角化例程 (bidiagonalization routine) 和奇异值分解（singular value decomposition, SVD）。

mcsolverDN 库主要面向 LAPACK 中广受欢迎的计算稠密型 (computationally-intensive) 例程，而且提供了与 LAPACK 兼容的 API。用户可以使用 mcsolverDN 加速这些耗时的例程，其他例程继续在 LAPACK 中，无需对现有代码进行重大修改。

1.2 命名约定 (Naming Conventions)

mcsolverDN 库提供两个不同的 API：`legacy` 和 `generic`。

传统 API 中的函数支持 `float`, `double`, `mcComplex` 和 `mcDoubleComplex` 这些数据类型。传统 API 的命名约定如下：

```
mcsolverDn<t><operation>
```

<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。<operation> 可以是 Cholesky 分解 (potrf), 使用部分主元消元法的 LU 分解 (getrf), QR 分解 (geqrf) 和 Bunch-Kaufman 分解 (sytrf)。

通用 API 中的函数为每个例程提供了单一的入口点，并支持使用 64 位整数定义矩阵和向量维度。通用 API 的命名约定与数据无关，并遵循以下规则：

```
mcsolverDn<t><operation>
```

<operation> 可以是 Cholesky 分解 (potrf)，使用部分主元消元法的 LU 分解 (getrf) 和 QR 分解 (geqrf)。

1.3 异步执行 (Asynchronous Execution)

mcsolver 库函数倾向于尽可能保持异步执行。开发人员可以始终使用 mcDeviceSynchronize() 函数来确保特定的 mcSOLVER 库例程已执行完成。

开发人员还可以通过 mcMemcpy() 例程，分别使用 mcMemcpyDeviceToHost 和 mcMemcpyHostToDevice 参数，将数据从设备复制到主机，或者从主机复制到设备。在这种情况下，不需要添加对 mcDeviceSynchronize() 的调用，因为使用上述参数调用 mcMemcpy() 是阻塞的，只有当结果在主机上准备好后才会完成。

1.4 库属性

libraryPropertyType 数据类型是库属性数据类型的枚举。（例如 MXMACA 版本 X.Y.Z 会产生 MAJOR_VERSION=X, MINOR_VERSION=Y, PATCH_LEVEL=Z）

```
typedef enum libraryPropertyType_t
{
    MAJOR_VERSION,
    MINOR_VERSION,
    PATCH_LEVEL
} libraryPropertyType;
```

以下代码可以显示 mcSOLVER 库的版本。

```
int major=-1,minor=-1,patch=-1;
mcsolverGetProperty(MAJOR_VERSION, &major);
mcsolverGetProperty(MINOR_VERSION, &minor);
mcsolverGetProperty(PATCH_LEVEL, &patch);
```

2 使用 mcSOLVER API

2.1 概述

本章介绍如何使用 mcSOLVER 库的 API。本章不是 mcSOLVER API 数据类型和函数的参考，这些内容将在后续章节中提供。

2.1.1 安装 mcSOLVER

mcSOLVER 是与 MXMACA 工具包一起发布和安装的库。MXMACA 工具包的安装请参见《曦云系列通用计算 GPU 快速上手指南》。在安装 MXMACA 工具包之后，请确保设置了环境变量 MACA_PATH。

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

mcSOLVER API 相关的文件如下所示。

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

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

在使用 mcSOLVER 库编译代码之前，请确保将环境变量 ISU_FASTMODEL 设置为 1。

```
export ISU_FASTMODEL=1
```

2.1.2 标量参数

在 mcSOLVER API 中，标量参数可以通过引用在主机上传递。

2.1.3 使用流进行并行处理

如果应用程序执行多个小型独立计算，或者在计算过程中进行数据传输，那么可以使用 MC 流来并行处理这些任务。

应用程序可以为每个任务关联一个流。为了实现任务之间的计算重叠，开发人员应该：

1. 使用函数 `mcStreamCreate()` 创建 MC 流。
2. 在调用实际的 mcsolverDN 例程之前，为每个独立的 mcsolver 库例程设置要使用的流，例如调用 `mcsolverDnSetStream()` 。

如果可能，在单独的流中执行的计算将在 GPU 上自动重叠。当单个任务执行的计算量相对较小，且不足以用工作填满 GPU，或者当数据传输可以与计算并行执行时，这种方法特别有用。

2.1.4 info 的约定

每个 LAPACK 例程返回一个 info，指示无效参数的位置。如果 info = -i，则第 i 个参数无效。为了与 LAPACK 从 1 开始的约定保持一致，mcsolver 不会将无效的 handle 报告到 info 中。相反，对于无效的 handle，mcsolver 返回 mcsolver_STATUS_NOT_INITIALIZED。

2.1.5 _bufferSize 的使用

mcsolver 库内部没有 mcMalloc 函数，用户必须明确地分配设备工作空间。例程 xyz_bufferSize 用于查询例程 xyz 的工作空间大小，例如 xyz = potrf。为了简化 API，在 xyz_bufferSize 中，几乎完全遵循了 xyz 的签名，即使它只依赖于某些参数，例如，不使用设备指针来决定工作空间的大小。在大多数情况下，在准备由设备指针指向的实际设备数据或分配设备指针之前，应在开始时调用 xyz_bufferSize。在这种情况下，用户可以将空指针传递给 xyz_bufferSize，而不会破坏功能性。

2.2 mcSOLVER 类型参考

2.2.1 mcsolverDN 类型

支持 float，double，mcComplex 和 mcDoubleComplex 数据类型。前两个是标准 C 数据类型，而后两个是从 mcComplex.h 导出的。此外，mcsolverDN 使用一些 mcBLAS 中常见的类型。

2.2.1.1 mcsolverDnHandle_t

这是一个指向不透明 mcsolverDN 上下文的指针类型，用户必须通过调用 mcsolverDnCreate() 初始化它，然后才能调用任何其他库函数。未初始化的句柄将导致意外行为，包括 mcsolverDN 崩溃。由 mcsolverDnCreate() 创建和返回的句柄必须传递给每个 mcsolverDN 函数。

2.2.1.2 mcblasFillMode_t

该类型指示稠密矩阵的哪个部分（下三角部分或上三角部分）被填充了，并因此指示函数应该使用哪个部分。其值对应于 Fortran 字符 L 或 l（下三角）和 U 或 u（上三角），这些字符通常被用作旧版 BLAS 实现的参数。

值	含义
MCBLAS_FILL_MODE_LOWER	填充了矩阵的下三角部分
MCBLAS_FILL_MODE_UPPER	填充了矩阵的上三角部分

2.2.1.3 mcblasOperation_t

mcblasOperation_t 类型指示需要对稠密矩阵执行哪种操作。其值对应于 Fortran 字符 'N' 或 'n'（不转置），'T' 或 't'（转置）和 'C' 或 'c'（共轭转置），这些值通常用作旧版 BLAS 实现的参数。

值	含义
MCBLAS_OP_N	选择非转置操作
MCBLAS_OP_T	选择转置操作
MCBLAS_OP_C	选择共轭转置操作

2.2.1.4 mcsolverEigType_t

mcsolverEigType_t 类型指示求解器计算的特征值的类型。其值对应于 Fortran 整数 1 ($A*x = \lambda*B*x$)，2 ($A*B*x = \lambda*x$)，3 ($B*A*x = \lambda*x$)，这些值通常用作旧版 LAPACK 实现的参数。

值	含义
MCSOLVER_EIG_TYPE_1	A*x = lambda*B*x
MCSOLVER_EIG_TYPE_2	A*B*x = lambda*x
MCSOLVER_EIG_TYPE_3	B*A*x = lambda*x

2.2.1.5 mcsolverEigMode_t

mcsolverEigMode_t 类型指示是否计算特征向量。其值对应于 Fortran 字符 'N' （仅计算特征值），'V' （同时计算特征值和特征向量），这些值通常用作旧版 LAPACK 实现的参数。

值	含义
MCSOLVER_EIG_MODE_NOVECTOR	仅计算特征值
MCSOLVER_EIG_MODE_VECTOR	同时计算特征值和特征向量

2.2.1.6 mcsolverDnFunction_t

mcsolverDnFunction_t 类型指示哪个例程需要由 mcsolverDnSetAdvOptions() 配置。对应于例程 Getrf 的值 mcsolverDN_GETRF。

值	含义
mcsolverDN_GETRF	对应于例程 Getrf 的值

2.2.1.7 mcsolverAlgMode_t

mcsolverAlgMode_t 类型指示 mcsolverDnSetAdvOptions() 选用的算法。每个例程支持的算法集详细描述在相应例程的文档中。

默认算法是 mcsolver_ALG_0。用户还可以提供 NULL 以使用默认算法。

2.2.1.8 mcsolverStatus_t

这是库函数返回的状态类型，它可以具有以下值。

值	含义
MCSOLVER_STATUS_SUCCESS	操作成功完成。
MCSOLVER_STATUS_NOT_INITIALIZED	mcsolver 库未进行初始化。通常是由于缺少先前调用、mcsolver 例程调用的 MC Runtime API 中的错误或硬件设置中的错误引起。 **解决方法**：在函数调用之前调用 mcsolverCreate()；并检查硬件、适当版本的驱动程序和 mcsolver 库是否正确安装。

下页继续

表 2.1 – 续上页

值	含义
MCSOLVER_STATUS_ALLOC_FAILED	mcsolver 库内部资源分配失败。通常是由 mcMalloc() 失败引起。 **解决方法**：在函数调用之前尽可能地释放先前分配的内存。
MCSOLVER_STATUS_INVALID_VALUE	向函数传递了一个不支持的值或参数（例如一个负的向量大小）。 **解决方法**：确保传递的所有参数都具有有效值。
MCSOLVER_STATUS_EXECUTION_FAILED	GPU 程序执行失败。这通常是由于 GPU 上内核启动失败引起的，其中可能有多个原因。 **解决方法**：检查硬件、适当版本的驱动程序和 mcsolver 库是否正确安装。
MCSOLVER_STATUS_INTERNAL_ERROR	内部 mcsolver 操作失败。通常是由于 mcMemcpyAsync() 失败引起。 **解决方法**：检查硬件、适当版本的驱动程序和 mcsolver 库是否正确安装。还要检查参数传递给例程的内存在例程完成之前是否未被释放。
MCSOLVER_STATUS_MATRIX_TYPE_NOT_SUPPORTED	该函数不支持矩阵类型。通常是由于将无效的矩阵描述符传递给函数引起的。 **解决方法**：检查 descrA 中的字段是否设置正确。

2.3 mcSOLVER 格式参考

2.3.1 向量（稠密）格式

假设向量在内存中线性存储。例如，向量

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

表示为

$$x = \begin{pmatrix} x_1 & x_2 & \dots & x_n \end{pmatrix}$$

2.3.2 矩阵（稠密）格式

假设稠密矩阵以列主序（column-major order）存储在内存中。可以使用原始矩阵的前导维度访问子矩阵（sub-matrix）。例如，`m*n` (sub-matrix)

$$\begin{pmatrix} a_{1,1} & \dots & a_{1,n} \\ a_{2,1} & \dots & a_{2,n} \\ \vdots & & \vdots \\ a_{m,1} & \dots & a_{m,n} \end{pmatrix}$$

表示为

$$\begin{pmatrix} a_{1,1} & \dots & a_{1,n} \\ a_{2,1} & \dots & a_{2,n} \\ \vdots & & \vdots \\ a_{m,1} & \dots & a_{m,n} \\ \vdots & & \vdots \\ a_{lda,1} & \dots & a_{lda,n} \end{pmatrix}$$

其中元素线性排列在内存中

$$(a_{1,1}, a_{2,1}, \dots, a_{m,1}, \dots, a_{lda,1}, \dots, a_{1,n}, a_{2,n}, \dots, a_{m,n}, \dots, a_{lda,n})$$

其中 `lda >= m` 是 `A` 的前导维度。

2.4 mcsolverDN：稠密 LAPACK 函数参考

这个部分描述了 mcsolverDN 提供稠密 LAPACK 函数子集的 API。

2.4.1 mcsolverDN 辅助函数参考

这个部分描述了 mcsolverDN 的辅助函数。

2.4.1.1 mcsolverDnCreate()

这个函数初始化了 mcsolverDN 库并在 mcsolverDN 上下文中创建一个句柄。在调用任何其他 mcsolverDN API 函数之前，必须先调用这个函数。它分配了访问 GPU 所需的硬件资源。

参数	内存	In/out	含义
`handle`	`host`	`output`	指针，指向 mcsolverDN 上下文中的句柄。

返回状态

值	含义
`MCSOLVER_STATUS_SUCCESS`	初始化成功。
`MCSOLVER_STATUS_NOT_INITIALIZED`	MC Runtime 初始化失败。
`MCSOLVER_STATUS_ALLOC_FAILED`	无法分配资源。

2.4.1.2 mcsolverDnDestroy()

这个函数释放 mcsolverDN 库使用的 CPU 端资源。

参数	内存	In/out	含义
`handle`	`host`	`input`	指针，指向 mcsolverDN 上下文中的句柄。

返回状态

值	含义
`MCSOLVER_STATUS_SUCCESS`	初始化成功。
`MCSOLVER_STATUS_NOT_INITIALIZED`	MC Runtime 初始化失败。

2.4.1.3 mcsolverDnSetStream()

```
mcsolverStatus_t
mcsolverDnSetStream(mcsolverDnHandle_t handle, mcStream_t streamId)
```

这个函数设置用于执行 mcsolverDN 库例程的流。

参数	内存	In/out	含义
handle	host	input	指针，指向 mcsolverDN 上下文中的句柄。
streamId	host	input	库使用的流

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。
MCSOLVER_STATUS_NOT_INITIALIZED	库未进行初始化。

2.4.1.4 mcsolverDnGetStream()

这个函数设置用于执行 mcsolverDN 库例程的流。

参数	内存	In/out	含义
handle	host	input	指针，指向 mcsolverDN 上下文中的句柄。
streamId	host	output	库使用的流

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。
MCSOLVER_STATUS_NOT_INITIALIZED	库未进行初始化。

2.4.1.5 mcsolverDnCreateSyevjInfo()

这个函数创建并初始化 syevj 、syevjBatched 和 sygvj 结构体，并将其设置为默认值。

参数	内存	In/out	含义
info	host	output	指针，指向 syevj 结构体。

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。
MCSOLVER_STATUS_NOT_INITIALIZED	库未进行初始化。

2.4.1.6 mcsolverDnDestroySyevjInfo()

这个函数销毁并释放了结构体所需的任何内存。

参数	内存	In/out	含义
info	host	output	syevj 结构体。

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。

2.4.1.7 mcsolverDnXsyevjSetTolerance()

这个函数配置 syevj 的容差。

参数	内存	In/out	含义
info	host	in/out	指针，指向 syevj 结构体。
tolerance	host	input	数值特征值的精度。

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。

2.4.1.8 mcsolverDnXsyevjSetMaxSweeps()

这个函数配置在 syevj 中扫描的最大次数。默认值为

参数	内存	In/out	含义
info	host	in/out	指针，指向 syevj 结构体。
max_sweeps	host	input	扫描的最大次数。

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。

2.4.1.9 mcsolverDnXsyevjSetSortEig()

如果 sort_eig 为零，则特征值未排序。这个函数仅适用于 syejvBatched。syevj 和 sygvj 总是按升序对特征值排序。默认地，总是按升序对特征值排序。

参数	内存	In/out	含义
info	host	in/out	指针，指向 syevj 结构体。
sort_eig	host	input	如果 sort_eig 为零，则特征值未排序。

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。

2.4.1.10 mcsolverDnXsyevjGetResidual()

```
mcsolverStatus_t
mcsolverDnXsyevjGetResidual(
    mcsolverDnHandle_t handle,
    syevjInfo_t info,
    double *residual)
```

这个函数反映了 syevj 或 sygvj 的残差。该函数不支持 syevjBatched。如果用户在 syevjBatched 之后调用这个函数，则会返回错误 mcsolver_STATUS_NOT_SUPPORTED。

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库的上下文句柄。
info	host	input	指针，指向 syevj 结构体。
residual	host	output	syevj 的残差。

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。
MCSOLVER_STATUS_NOT_SUPPORTED	不支持 batched 版本。

2.4.1.11 mcsolverDnXsyevjGetSweeps()

```
mcsolverStatus_t
mcsolverDnXsyevjGetSweeps(
    mcsolverDnHandle_t handle,
    syevjInfo_t info,
    int *executed_sweeps)
```

这个函数反映了 syevj 或 sygvj 执行扫描的次数。该函数不支持 syejvBatched 。如果用户在 syevjBatched 之后调用这个函数，则会返回错误 mcsolver_STATUS_NOT_SUPPORTED 。

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库的上下文句柄。
info	host	input	指针，指向 syevj 结构体。
executed_sweeps	host	output	执行扫描的次数。

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。
MCSOLVER_STATUS_NOT_SUPPORTED	不支持 batched 版本。

2.4.1.12 mcsolverDnCreateGesvdjInfo()

```
mcsolverStatus_t
mcsolverDnCreateGesvdjInfo(
    gesvdjInfo_t *info);
```

这个函数初始化了 gesvdj 和 gesvdjBatched ，并将其设置为默认值。

参数	内存	In/out	含义
info	host	input	指针，指向 gesvdj 结构体。

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。
MCSOLVER_STATUS_ALLOC_FAILED	无法分配资源。

2.4.1.13 mcsolverDnDestroyGesvdjInfo()

这个函数销毁并释放了结构体所需的任何内存。

参数	内存	In/out	含义
info	host	output	gesvdj 结构体

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。

2.4.1.14 mcsolverdnxgesvdjsettolerance()

这个函数配置 gesvdj 的容差。

参数	内存	In/out	含义
info	host	in/out	指针，指向 gesvdj 结构体。
tolerance	host	input	数值特征值的精度。

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。

2.4.1.15 mcsolverDnXgesvdjSetMaxSweeps()

这个函数配置在 gesvdj 中扫描的最大次数。默认值为

参数	内存	In/out	含义
info	host	in/out	指针，指向 gesvdj 结构体。
max_sweeps	host	input	扫描的最大次数。

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。

2.4.1.16 mcsolverDnXgesvdjSetSortEig()

如果 sort_svd 为零，则奇异值未排序。这个函数仅适用于 gesvdjBatched。gesvdj 总是按降序对奇异值排序。默认地，总是按降序对奇异值排序。

参数	内存	In/out	含义
info	host	in/out	指针，指向 gesvdj 结构体。
sort_svd	host	input	如果 sort_svd 为零，则奇异值未排序。

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。

2.4.1.17 mcsolverDnXgesvdjGetResidual()

```
mcsolverStatus_t
mcsolverDnXgesvdjGetResidual(
    mcsolverDnHandle_t handle,
    gesvdjInfo_t info,
    double *residual)
```

这个函数反映了 gesvdj 的残差。该函数不支持 gesvdjBatched。如果用户在 syevjBatched 之后调用这个函数，则会返回错误 mcsolver_STATUS_NOT_SUPPORTED。

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库的上下文句柄。
info	host	input	指针，指向 gesvdj 结构体。
residual	host	output	gesvdj 的残差。

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。
MCSOLVER_STATUS_NOT_SUPPORTED	不支持 batched 版本。

2.4.1.18 mcsolverDnXgesvdjGetSweeps()

```
mcsolverStatus_t
mcsolverDnXgesvdjGetSweeps(
    mcsolverDnHandle_t handle,
    gesvdjInfo_t info,
    int *executed_sweeps)
```

这个函数反映了 gesvdj 执行扫描的次数。该函数不支持 syejvBatched。如果用户在 syevjBatched 之后调用这个函数，则会返回错误 mcsolver_STATUS_NOT_SUPPORTED。

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库的上下文句柄。
info	host	input	指针，指向 gesvdj 结构体。
executed_sweeps	host	output	执行扫描的次数。

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。
MCSOLVER_STATUS_NOT_SUPPORTED	不支持 batched 版本。

2.4.1.19 mcsolverDnCreateParams()

```
mcsolverStatus_t
mcsolverDnCreateParams(
    mcsolverDnParams_t *params);
```

这个函数初始化了 64-bit API 结构，并将其设置为默认值。

参数	内存	In/out	含义
params	host	in/out	指向 64-bit API 结构的指针

返回状态

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。
MCSOLVER_STATUS_ALLOC_FAILED	无法分配资源。

2.4.1.20 mcsolverDnDestroyParams()

```
mcsolverStatus_t
mcsolverDnDestroyParams(
    mcsolverDnParams_t params);
```

这个函数销毁并释放了结构所需的任何内存。

参数	内存	In/out	含义
params	host	in/out	指向 64-bit API 结构的指针

返回状态

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。

2.4.1.21 mcsolverDnSetAdvOptions()

```
mcsolverStatus_t
mcsolverDnSetAdvOptions (
    mcsolverDnParams_t params,
    mcsolverDnFunction_t function,
    mcsolverAlgMode_t algo   );
```

该函数配置了 function 的算法 algo，它是一个 64-bit API 例程。

参数	内存	In/out	含义
params	host	in/out	指向 64-bit API 结构体的指针。
funtion	host	input	需要配置的函数。
algo	host	input	需要配置的算法。

返回状态

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。
MCSOLVER_STATUS_INVALID_VALUE	函数和算法的组合错误。

2.4.2 稠密线性求解器参考（legacy）

本节描述了 mcsolverDN 的线性求解器 API，包括 Cholesky 分解、带有部分主元的 LU 分解、QR 分解和 Bunch-Kaufman（LDLT）分解。

2.4.2.1 mcsolverDn<t>potrf()

这些辅助函数计算所需的工作缓冲区大小。

```
mcsolverStatus_t
mcsolverDnSpotrf_bufferSize(mcsolverDnHandle_t handle,
                mcblasFillMode_t uplo,
                int n,
                float *A,
                int lda,
                int *Lwork );

mcsolverStatus_t
mcsolverDnDpotrf_bufferSize(mcsolveDnHandle_t handle,
                mcblasFillMode_t uplo,
                int n,
                double *A,
```

(下页继续)

(续上页)

```
                int lda,
                int *Lwork );

mcsolverStatus_t
mcsolverDnCpotrf_bufferSize(mcsolverDnHandle_t handle,
                mcblasFillMode_t uplo,
                int n,
                mcComplex *A,
                int lda,
                int *Lwork );

mcsolverStatus_t
mcsolverDnZpotrf_bufferSize(mcsolverDnHandle_t handle,
                mcblasFillMode_t uplo,
                int n,
                mcDoubleComplex *A,
                int lda,
                int *Lwork);
```

S 和 D 数据类型分别表示单精度实数和双精度实数。

```
mcsolverStatus_t
mcsolverDnSpotrf(mcsolverDnHandle_t handle,
            mcblasFillMode_t uplo,
            int n,
            float *A,
            int lda,
            float *Workspace,
            int Lwork,
            int *devInfo );

mcsolverStatus_t
mcsolverDnDpotrf(mcsolverDnHandle_t handle,
            mcblasFillMode_t uplo,
            int n,
            double *A,
            int lda,
            double *Workspace,
            int Lwork,
            int *devInfo );
```

C 和 Z 数据类型分别表示单精度复数和双精度复数。

```
mcsolverStatus_t
mcsolverDnCpotrf(mcsolverDnHandle_t handle,
            mcblasFillMode_t uplo,
            int n,
            mcComplex *A,
            int lda,
            mcComplex *Workspace,
            int Lwork,
            int *devInfo );

mcsolverStatus_t
mcsolverDnZpotrf(mcsolverDnHandle_t handle,
            mcblasFillMode_t uplo,
            int n,
```

(下页继续)

(续上页)

```
            mcDoubleComplex *A,
            int lda,
            mcDoubleComplex *Workspace,
            int Lwork,
            int *devInfo );
```

这个函数计算一个厄米特正定矩阵的 Cholesky 分解。

A 是一个 n×n 的厄米特矩阵，但只有下三角或上三角部分是有意义。输入参数 uplo 指示函数使用矩阵的哪一部分，其他部分保持不变。

如果输入参数 uplo 是 MCBLAS_FILL_MODE_LOWER，则只处理矩阵 A 的下三角部分，并用下三角的 Cholesky 分解因子 L 替换原有的下三角部分。

$$A = L * L^H$$

如果输入参数 uplo 是 MCBLAS_FILL_MODE_UPPER，则只处理矩阵 A 的上三角部分，并用上三角的 Cholesky 分解因子 U 替换原有的上三角部分。

$$A = U^H * U$$

用户必须提供输入参数 Workspace 所指向的工作空间。输入参数 Lwork 表示工作空间的大小，通过调用函数 potrf_bufferSize() 返回。

如果 Cholesky 分解失败，即矩阵 A 的某个前导子矩阵不是正定的，或者说 L 或 U 的某个对角线元素不是实数，那么输出参数 devInfo 将指示 A 的最小且不是正定的前导子矩阵。

如果输出参数 devInfo = -i （小于零），表示第 i 个参数出现了错误（不包含句柄）。

API of potrf

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库的上下文句柄。
uplo	host	input	指出存储矩阵 A 的下三角还是上三角。另一部分没有引用。
n	host	input	矩阵 A 的行数和列数。
A	device	input	维度为 lda * n 的 <type> 数组，其中 lda 小于 max(1,n)。
lda	host	input	用于存储矩阵 A 的二维数组前导维度。
Workspace	device	in/out	工作空间，是一个类型为 <type> 的数组，其大小为 Lwork。
Lwork	host	input	由函数 potrf_bufferSize 返回的工作空间的大小。
devInfo	device	output	• 当 devInfo = 0 时，表示 Cholesky 分解成功。 • 当 devInfo = -i 时，表示第 i 个参数有问题（不包含句柄）。 • 当 devInfo = i 时，表示第 i 个前导子矩阵不是正定的。

返回状态

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。
MCSOLVER_STATUS_IRS_INFOS_NOT_INITIALIZED	库未进行初始化
MCSOLVER_STATUS_INVALID_VALUE	传递的参数无效 (n<0 或 lda<max(1,n))。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.2.2 mcsolverDnPotrf()

下面的辅助函数可以计算所需的预分配缓冲区大小。

```
mcsolverStatus_t
mcsolverDnPotrf_bufferSize(
    mcsolverDnHandle_t handle,
    mcsolverDnParams_t params,
    mcblasFillMode_t uplo,
    int64_t n,
    macaDataType dataTypeA,
    const void *A,
    int64_t lda,
    macaDataType computeType,
    size_t *workspaceInBytes )
```

下面的例程：

```
mcsolverStatus_t
mcsolverDnPotrf(
    mcsolverDnHandle_t handle,
    mcsolverDnParams_t params,
    mcblasFillMode_t uplo,
    int64_t n,
    macaDataType dataTypeA,
    void *A,
    int64_t lda,
    macaDataType computeType,
    void *pBuffer,
    size_t workspaceInBytes,
    int *info )
```

使用通用 API 接口计算厄米特正定矩阵的 Cholesky 分解。

`A` 是一个 `n×n` 的厄米特矩阵，但只有下三角或上三角部分是有意义。输入参数 `uplo` 指示函数使用矩阵的哪一部分，其他部分保持不变。

如果输入参数 `uplo` 是 `MCBLAS_FILL_MODE_LOWER`，则只处理矩阵 `A` 的下三角部分，并用下三角的 Cholesky 分解因子 `L` 替换原有的下三角部分。

$$A = L * L^H$$

如果输入参数 `uplo` 是 `MCBLAS_FILL_MODE_UPPER`，则只处理矩阵 `A` 的上三角部分，并用上三角的 Cholesky 分解因子 `U` 替换原有的上三角部分。

$$A = U^H * U$$

如果 Cholesky 分解失败，即矩阵 `A` 的某个前导子矩阵不是正定的，或者说 `L` 或 `U` 的某个对角线元素不是实数，那么输出参数 `devInfo` 将指示 `A` 的最小且不是正定的前导子矩阵。

如果输出参数 `devInfo = -i`（小于零），表示第 i 个参数出现了错误（不包含句柄）。

目前，`mcsolverDnPotrf` 仅支持默认的算法

mcsolverDnPotrf 支持的算法表

`mcsolver_ALG_0` 或 `NULL`	默认算法

API of potrf

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库的上下文句柄。
params	host	input	由 mcsolverDnSetAdvOptions 收集的信息的结构体。
uplo	host	input	指出存储矩阵 A 的下三角还是上三角。另一部分没有引用。
n	host	input	矩阵 A 的行数和列数。
A	device	input	维度为 lda * n 的 <type> 数组，其中 lda 小于 max(1,n)。
dataTypeA	host	in	数组 A 的数据类型。
lda	host	input	用于存储矩阵 A 的二维数组前导维度。
computeType	host	in	计算的数据类型。
pBuffer	device	in/out	工作空间。大小为 workspaceInBytes 字节的 void 类型数组。
workspaceInBytes	host	input	由函数 potrf_bufferSize 返回的工作空间的大小。
info	device	output	• 如果 info = 0，Cholesky 分解成功。 • 如果 info = -i，表示第 i 个参数有误 (不包含句柄)。 • 如果 info = i，则阶数为 i 的前导子矩阵不是正定的。

通用 API 有两种不同的类型，dataTypeA 是矩阵 A 的数据类型，computeType 是操作的计算类型。mcsolverDnPotrf 仅支持以下四种组合。

数据类型和计算类型的有效组合

DataTypeA	ComputeType	Meaning
MACA_R_32F	MACA_R_32F	SPOTRF
MACA_R_64F	MACA_R_64F	DPOTRF
MACA_C_32F	MACA_C_32F	CPOTRF
MACA_C_64F	MACA_C_64F	ZPOTRF

返回状态

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。
MCSOLVER_STATUS_NOT_INITIALIZED	库未进行初始化
MCSOLVER_STATUS_INVALID_VALUE	传递的参数无效 (n<0 或 lda<max(1,n))。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.2.3 mcsolverDn<t>potrs()

```
mcsolverStatus_t
mcsolverDnSpotrs(mcsolverDnHandle_t handle,
            mcblasFillMode_t uplo,
            int n,
            int nrhs,
            const float *A,
            int lda,
            float *B,
            int ldb,
            int *devInfo);

mcsolverStatus_t
mcsolverDnDpotrs(mcsolverDnHandle_t handle,
            mcblasFillMode_t uplo,
```

(下页继续)

(续上页)

```
            int n,
            int nrhs,
            const double *A,
            int lda,
            double *B,
            int ldb,
            int *devInfo);

mcsolverStatus_t
mcsolverDnCpotrs(mcsolverDnHandle_t handle,
            mcblasFillMode_t uplo,
            int n,
            int nrhs,
            const mcComplex *A,
            int lda,
            mcComplex *B,
            int ldb,
            int *devInfo);

mcsolverStatus_t
mcsolverDnZpotrs(mcsolverDnHandle_t handle,
            mcblasFillMode_t uplo,
            int n,
            int nrhs,
            const mcDoubleComplex *A,
            int lda,
            mcDoubleComplex *B,
            int ldb,
            int *devInfo);
```

这个函数用于求解一个线性方程组。

$$A * X = B$$

A 是一个 n×n 的厄米特矩阵，但只有下三角或上三角部分是有意义。输入参数 uplo 指示函数使用矩阵的哪一部分，其他部分保持不变。

用户首先需要调用 potrf 函数对矩阵 A 进行 Cholesky 分解。

- 如果输入参数 uplo 为 MCBLAS_FILL_MODE_LOWER ，则 A 是下三角 Cholesky 分解因子 L 并满足 $A = L * L^H$。
- 如果输入参数 uplo 为 MCBLAS_FILL_MODE_UPPER ，则 A 是上三角 Cholesky 分解因子 U 并满足 $A = U^H * U$。

这个操作是就地进行的，即矩阵 X 会覆盖矩阵 B ，并且它们有相同的前导维度 ldb 。

如果输出参数 devInfo = -i （小于零），表示第 i 个参数出现了错误（不包含句柄）。

API of potrs

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库的上下文句柄。
uplo	host	input	指出存储矩阵 A 的下三角还是上三角。另一部分没有引用。
n	host	input	矩阵 A 的行数和列数。
nrhs	host	input	矩阵 X 和 B 的列数。
A	device	input	<type> 数组，维度为 lda * n，其中 lda 不小于 max(1,n)。 A 可以是下三角形的 Cholesky 因子 L，也可以是上三角形的 Cholesky 因子 U。

下页继续

表 2.2 – 续上页

参数	内存	In/out	含义
lda	host	input	用于存储矩阵 A 的二维数组前导维度。
B	device	in/out	维度为 ldb * nrhs 的 <type> 数组。其中，ldb 至少要大于等于 max(1, n)。作为输入时，矩阵 B 是右端矩阵。作为输出时，矩阵 B 是解的矩阵。
info	device	output	• 如果 info = 0，Cholesky 分解成功。 • 如果 info = -i，表示第 i 个参数有误 (不包含句柄)。 • 如果 info = i，则阶数为 i 的前导子矩阵不是正定的。

返回状态

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。
MCSOLVER_STATUS_NOT_INITIALIZED	库未进行初始化
MCSOLVER_STATUS_INVALID_VALUE	传递的参数无效 (n<0 或 lda<max(1,n))。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.2.4 mcsolverDnPotrs()

```
mcsolverStatus_t
mcsolverDnPotrs(
    mcsolverDnHandle_t handle,
    mcsolverDnParams_t params,
    mcblasFillMode_t uplo,
    int64_t n,
    int64_t nrhs,
    macaDataType dataTypeA,
    const void *A,
    int64_t lda,
    macaDataType dataTypeB,
    void *B,
    int64_t ldb,
    int *info)
```

这个函数用来求解线性方程组

$$A * X = B$$

A 是一个 n×n 的厄米特矩阵，在使用通用 API 接口时，只有下三角部分或上三角部分是有意义的。输入参数 uplo 指示函数使用矩阵的哪一部分，其他部分保持不变。

用户首先需要调用 mcsolverDnPotrf 函数对矩阵 A 进行分解 - 输入参数 uplo 为 MCBLAS_FILL_MODE_LOWER，则 A 是下三角 Cholesky 分解因子 L 并满足 $A = L * L^H$。- 如果输入参数 uplo 为 MCBLAS_FILL_MODE_UPPER，则 A 是上三角 Cholesky 分解因子 U 并满足 $A = U^H * U$。

这个操作是就地进行的，即矩阵 X 会覆盖矩阵 B，并且它们有相同的前导维度 ldb。

如果输出参数 devInfo = -i （小于零），表示第 i 个参数出现了错误（不包含句柄）。

目前，mcsolverDnPotrs 仅支持默认算法。

mcsolverDnPotrs 支持的算法表

mcsolver_ALG_0 或 NULL	默认算法

API of potrs

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库的上下文句柄。
params	host	input	由 mcsolverDnSetAdvOptions 收集的信息的结构体。
uplo	host	input	指出存储矩阵 A 的下三角还是上三角。另一部分没有引用。
n	host	input	矩阵 A 的行数和列数。
nrhs	host	input	矩阵 X 和 B 的列数。
A	device	input	维度为 lda * n 的 <type> 数组，其中 lda 小于 max(1,n)。
dataTypeA	host	in	数组 A 的数据类型。
lda	host	input	用于存储矩阵 A 的二维数组前导维度。
dataTypeB	host	in	数组 B 的数据类型。
computeType	host	in	计算的数据类型。
pBuffer	device	in/out	工作空间。大小为 workspaceInBytes 字节的 void 类型数组。
B	device	input	维度为 ldb * nrhs 的 <type> 数组。其中，ldb 至少要大于等于 max(1, n)。作为输入时，矩阵 B 是右端矩阵。作为输出时，矩阵 B 是解的矩阵。
info	device	output	• 如果 info = 0，Cholesky 分解成功。 • 如果 info = -i，表示第 i 个参数有误 (不包含句柄)。 • 如果 info = i，则阶数为 i 的前导子矩阵不是正定的。

通用 API 具有两种不同的类型：dataTypeA 是矩阵 A 的数据类型，dataTypeB 是矩阵 B 的数据类型。mcsolverDnPotrs 只支持以下四种组合。

数据类型和计算类型的有效组合

dataTypeA	dataTypeB	Meaning
MACA_R_32F	MACA_R_32F	SPOTRS
MACA_R_64F	MACA_R_64F	DPOTRS
MACA_C_32F	MACA_C_32F	CPOTRS
MACA_C_64F	MACA_C_64F	ZPOTRS

返回状态

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。
MCSOLVER_STATUS_NOT_INITIALIZED	库未进行初始化
MCSOLVER_STATUS_INVALID_VALUE	传递的参数无效 (n<0 或 lda<max(1,n))。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.2.5 mcsolverDn<t>potri()

这些辅助函数用于计算所需的工作缓冲区大小。

```
mcsolverStatus_t
mcsolverDnSpotri_bufferSize(mcsolverDnHandle_t handle,
                mcblasFillMode_t uplo,
                int n,
                float *A,
                int lda,
                int *Lwork );

mcsolverStatus_t
```

(下页继续)

(续上页)

```
mcsolverDnDpotri_bufferSize(mcsolveDnHandle_t handle,
                mcblasFillMode_t uplo,
                int n,
                double *A,
                int lda,
                int *Lwork );

mcsolverStatus_t
mcsolverDnCpotri_bufferSize(mcsolverDnHandle_t handle,
                mcblasFillMode_t uplo,
                int n,
                mcComplex *A,
                int lda,
                int *Lwork );

mcsolverStatus_t
mcsolverDnZpotri_bufferSize(mcsolverDnHandle_t handle,
                mcblasFillMode_t uplo,
                int n,
                mcDoubleComplex *A,
                int lda,
                int *Lwork);
```

S 和 D 数据类型分别表示单精度实数和双精度实数。

```
mcsolverStatus_t
mcsolverDnSpotri(mcsolverDnHandle_t handle,
            mcblasFillMode_t uplo,
            int n,
            float *A,
            int lda,
            float *Workspace,
            int Lwork,
            int *devInfo );

mcsolverStatus_t
mcsolverDnDpotri(mcsolverDnHandle_t handle,
            mcblasFillMode_t uplo,
            int n,
            double *A,
            int lda,
            double *Workspace,
            int Lwork,
            int *devInfo );
```

C 和 Z 数据类型分别表示单精度复数和双精度复数。

```
mcsolverStatus_t
mcsolverDnCpotri(mcsolverDnHandle_t handle,
            mcblasFillMode_t uplo,
            int n,
            mcComplex *A,
            int lda,
            mcComplex *Workspace,
            int Lwork,
            int *devInfo );
```

(下页继续)

(续上页)

```
mcsolverStatus_t
mcsolverDnZpotri(mcsolverDnHandle_t handle,
            mcblasFillMode_t uplo,
            int n,
            mcDoubleComplex *A,
            int lda,
            mcDoubleComplex *Workspace,
            int Lwork,
            int *devInfo );
```

这个函数使用由 potrf() 计算得到的 Cholesky 分解，来计算一个正定矩阵 A 的逆矩阵。

A 是一个 n×n 的矩阵，其中包含了由 Cholesky 分解得到的下三角因子 L 或上三角因子 U。只有下三角部分或上三角部分是有意义的，输入参数 uplo 指示函数使用矩阵的哪一部分，其他部分保持不变。

如果输入参数 uplo 的值为 MCBLAS_FILL_MODE_LOWER，那么只会处理矩阵 A 的下三角部分，并用 A 的下三角部分的逆矩阵替换原始矩阵的下三角部分。

如果输入参数 uplo 的值为 MCBLAS_FILL_MODE_UPPER，那么只会处理矩阵 A 的上三角部分，并用 A 的上三角部分的逆矩阵替换原始矩阵的上三角部分。

用户需要提供一个工作空间，该工作空间由输入参数 Workspace 指向。同时，用户还需要提供输入参数 Lwork，它表示工作空间的大小。这个大小是通过调用函数 potri_bufferSize() 来获取的，它用于计算所需的工作空间大小。

如果计算逆矩阵失败，即某些 L 或 U 的前导子矩阵为零，那么输出参数 devInfo 将指示 L 或 U 中最小的前导子矩阵，并且该子矩阵不是正定的。

如果输出参数 devInfo = -i （小于零），表示第 i 个参数出现了错误（不包含句柄）。

API of potri

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库的上下文句柄。
uplo	host	input	指出存储矩阵 A 的下三角还是上三角。另一部分没有引用。
n	host	input	矩阵 A 的行数和列数。
A	device	input	<type> 数组，维度为 lda * n，其中 lda 不小于 max(1,n)。A 可以是下三角形的 Cholesky 因子 L，也可以是上三角形的 Cholesky 因子 U。
lda	host	input	用于存储矩阵 A 的二维数组前导维度。
Workspace	host	input	工作空间。尺寸为 Lwork 的 <type> 数组。
Lwork	host	input	工作空间尺寸。由 potri_bufferSize 返回。
info	device	output	• 如果 info = 0，Cholesky 分解成功。 • 如果 info = -i，表示第 i 个参数有误 (不包含句柄)。 • 如果 info = i，则阶数为 i 的前导子矩阵不是正定的。

返回状态

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。
MCSOLVER_STATUS_NOT_INITIALIZED	库未进行初始化
MCSOLVER_STATUS_INVALID_VALUE	传递的参数无效 (n<0 或 lda<max(1,n))。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.2.6 mcsolverDn<t>getrf()

这些辅助函数计算所需的工作缓冲区的大小。

```
mcsolverStatus_t
mcsolverDnSgetrf_bufferSize(mcsolverDnHandle_t handle,
                            int m,
                            int n,
                            float *A,
                            int lda,
                            int *Lwork );

mcsolverStatus_t
mcsolverDnDgetrf_bufferSize(mcsolverDnHandle_t handle,
                            int m,
                            int n,
                            double *A,
                            int lda,
                            int *Lwork );

mcsolverStatus_t
mcsolverDnCgetrf_bufferSize(mcsolverDnHandle_t handle,
                            int m,
                            int n,
                            mcComplex *A,
                            int lda,
                            int *Lwork );

mcsolverStatus_t
mcsolverDnZgetrf_bufferSize(mcsolverDnHandle_t handle,
                            int m,
                            int n,
                            mcDoubleComplex *A,
                            int lda,
                            int *Lwork );
```

S 和 D 数据类型分别表示单精度实数和双精度实数。

```
mcsolverStatus_t
mcsolverDnSgetrf(mcsolverDnHandle_t handle,
                 int m,
                 int n,
                 float *A,
                 int lda,
                 float *Workspace,
                 int *devIpiv,
                 int *devInfo );

mcsolverStatus_t
mcsolverDnDgetrf(mcsolverDnHandle_t handle,
                 int m,
                 int n,
                 double *A,
                 int lda,
                 double *Workspace,
                 int *devIpiv,
                 int *devInfo );
```

C 和 Z 数据类型分别表示单精度复数和双精度复数。

```
mcsolverStatus_t
mcsolverDnCgetrf(mcsolverDnHandle_t handle,
            int m,
            int n,
            mcComplex *A,
            int lda,
            mcComplex *Workspace,
            int *devIpiv,
            int *devInfo );

mcsolverStatus_t
mcsolverDnZgetrf(mcsolverDnHandle_t handle,
            int m,
            int n,
            mcDoubleComplex *A,
            int lda,
            mcDoubleComplex *Workspace,
            int *devIpiv,
            int *devInfo );
```

该函数计算一个 m×n 矩阵的 LU 分解

$$P * A = L * U$$

其中 A 是一个 m×n 矩阵，P 是一个置换矩阵，L 是一个下三角矩阵且对角线上元素为 1，U 是一个上三角矩阵。

用户需要提供由输入参数 Workspace 指向的工作空间。输入参数 Lwork 表示工作空间的大小，并且可以通过 getrf_bufferSize() 函数获得。

如果 LU 分解失败，即矩阵 A （或者 U ）奇异，则输出参数 devInfo=i 表示 U(i,i) = 0 。

如果输出参数 devInfo = -i （小于零），则表示第 i 个参数错误（不包含句柄）。

如果 devIpiv 为空，则不进行主元交换。此时分解结果为 A=L*U ，但数值稳定性较差。

无论 LU 分解失败与否，输出参数 devIpiv 包含主元交换的序列，第 i 行与第 devIpiv(i) 行进行了交换。

用户可以结合 getrf 和 getrs 函数完成线性求解器任务。

注解: getrf 使用了具有大小为 m*n 的大型工作空间的最快实现。用户可以通过 Getrf 和 mcsolverDnSetAdvOptions(params,mcsolverDN_GETRF,mcsolver_ALG_1) 选择工作空间最小的传统实现。

---

getrf API

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库的上下文句柄。
m	host	input	矩阵 A 的行数。
n	host	input	矩阵 A 的列数。
A	device	input	<type> 数组，维度为 lda * n，其中 lda 不小于 max(1,n)。A 可以是下三角形的 Cholesky 因子 L，也可以是上三角形的 Cholesky 因子 U。
lda	host	input	用于存储矩阵 A 的二维数组前导维度。
Workspace	device	input	工作空间，大小为 Lwork 的 <type> 数组。
devIpiv	device	output	大小至少为 min(m,n) 的数组，包含主元索引。
info	device	output	• 如果 info = 0，表示 Cholesky 分解成功。 • 如果 info = -i，表示第 i 个参数错误（不包含句柄）。 • 如果 info = i，则表示阶数为 i 的前导子矩阵不是正定的。

返回状态

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。
MCSOLVER_STATUS_NOT_INITIALIZED	库未进行初始化。
MCSOLVER_STATUS_INVALID_VALUE	传递的参数无效 (n<0 或 lda<max(1,n))。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.2.7 mcsolverDnGetrf()

下面的辅助函数可以计算所需的预分配缓冲区大小。

```
mcsolverStatus_t
mcsolverDnGetrf_bufferSize(
    mcsolverDnHandle_t handle,
    mcsolverDnParams_t params,
    int64_t m,
    int64_t n,
    macaDataType dataTypeA,
    const void *A,
    int64_t lda,
    macaDataType computeType,
    size_t *workspaceInBytes )
```

以下函数:

```
mcsolverStatus_t
mcsolverDnGetrf(
    mcsolverDnHandle_t handle,
    mcsolverDnParams_t params,
    int64_t m,
    int64_t n,
    macaDataType dataTypeA,
    void *A,
    int64_t lda,
    int64_t *ipiv,
    macaDataType computeType,
    void *pBuffer,
```

(下页继续)

(续上页)

```
    size_t workspaceInBytes,
    int *info )
```

计算一个 m x n 矩阵的 LU 分解。

$$P * A = L * U$$

其中 A 是一个 m×n 矩阵，P 是一个置换矩阵，L 是一个下三角矩阵且对角线上元素为 1，U 是一个使用通用 API 接口实现的上三角矩阵。

如果 LU 分解失败，即矩阵 A （或者 U ）奇异，则输出参数 info = -i 表示 U(i,i) = 0 。

如果输出参数 info = -i （小于零），则表示第 i 个参数错误（不包含句柄）。

如果 ipiv 为空，此时分解结果为 A=L*U ，但数值稳定性较差。

无论 LU 分解失败与否，输出参数 ipiv 包含主元交换的序列，第 i 行与第 ipiv(i) 行进行了交换。

用户需要提供由输入参数 pBuffer 指向的工作空间。输入参数 workspaceInBytes 是工作空间的字节大小，可以通过 mcsolverDnGetrf_bufferSize() 函数获得。

用户可以结合 mcsolverDnGetrf 和 mcsolverDnGetrs 函数完成线性求解器任务。

目前，mcsolverDnGetrf 支持两种算法。若要选择传统实现，用户需调用 mcsolverDnSetAdvOptions 函数设置。

mcsolverDnGetrf 支持的算法表格

MCSOLVER_ALG_0 或 NULL	默认算法。速度最快，需要一个较大的 m*n 元素工作空间
MCSOLVER_ALG_1	传统实现

mcsolverDnGetrf_bufferSize 和 mcsolverDnGetrf 的输入参数列表如下：

API of mcsolverDNGetrf

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库的上下文句柄。
params	host	input	由 mcsolverDnSetAdvOptions 收集的信息的结构体。
m	host	input	矩阵 A 的行数。
n	host	input	矩阵 A 的列数。
dataTypeA	host	in	数组 A 的数据类型。
A	device	in/out	<type> 数组，维度为 lda * n，其中 lda 不小于 max(1,n)。A 可以是下三角形的 Cholesky 因子 L，也可以是上三角形的 Cholesky 因子 U。
lda	host	input	用于存储矩阵 A 的二维数组前导维度。
ipiv	device	output	大小至少为 min(m,n) 的数组，包含主元索引。
computeType	host	in	计算的数据类型。
pBuffer	device	in/out	工作空间。大小为 workspaceInBytes 字节的 void 类型数组。
workspaceInBytes	host	input	pBuffer 的字节大小，由 mcsolverDnGetrf_bufferSize 返回。
info	device	output	• 如果 info = 0，表示 Cholesky 分解成功。 • 如果 info = -i，表示第 i 个参数错误（不包含句柄）。 • 如果 info = i，则表示阶数为 i 的前导子矩阵不是正定的。

通用 API 有两种不同的类型，dataTypeA 是矩阵 A 的数据类型，computeType 是操作的计算类型。mcsolverDnGetrf 仅支持以下四种组合。

数据类型和计算类型的有效组合

DataTypeA	ComputeType	Meaning
MACA_R_32F	MACA_R_32F	SGETRF
MACA_R_64F	MACA_R_64F	DGETRF
MACA_C_32F	MACA_C_32F	CGETRF
MACA_C_64F	MACA_C_64F	ZGETRF

返回状态

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。
MCSOLVER_STATUS_NOT_INITIALIZED	库未进行初始化。
MCSOLVER_STATUS_INVALID_VALUE	传递的参数无效 (n<0 或 lda<max(1,n))。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.2.8 mcsolverDn<t>getrs()

```
mcsolverStatus_t
mcsolverDnSgetrs(mcsolverDnHandle_t handle,
            mcblasOperation_t trans,
            int n,
            int nrhs,
            const float *A,
            int lda,
            const int *devIpiv,
            float *B,
            int ldb,
            int *devInfo );

mcsolverStatus_t
mcsolverDnDgetrs(mcsolverDnHandle_t handle,
            mcblasOperation_t trans,
            int n,
            int nrhs,
            const double *A,
            int lda,
            const int *devIpiv,
            double *B,
            int ldb,
            int *devInfo );

mcsolverStatus_t
mcsolverDnCgetrs(mcsolverDnHandle_t handle,
            mcblasOperation_t trans,
            int n,
            int nrhs,
            const mcComplex *A,
            int lda,
            const int *devIpiv,
            mcComplex *B,
            int ldb,
            int *devInfo );

mcsolverStatus_t
```

(下页继续)

(续上页)

```
mcsolverDnZgetrs(mcsolverDnHandle_t handle,
            mcblasOperation_t trans,
            int n,
            int nrhs,
            const mcDoubleComplex *A,
            int lda,
            const int *devIpiv,
            mcDoubleComplex *B,
            int ldb,
            int *devInfo );
```

这个函数解决了一组包含多个右端量的线性方程组

$$op(A) * X = B$$

其中 A 是一个 n×n 的矩阵，通过 getrf 进行了 LU 分解，即 A 的下三角部分是 L，上三角部分（包括对角线元素）是 U。B 是一个 n×nrhs 的右端量矩阵。

输入参数 devIpiv 是 getrf 的输出，它包含了用于对右端量进行置换的主元索引。

如果输出参数 devInfo = -i （小于零），则表示第 i 个参数错误（不包含句柄）。

用户可以结合 getrf 和 getrs 函数完成线性求解器任务。

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库的上下文句柄。
trans	host	input	操作 op(A) 的非转置或（共轭）转置。
n	host	input	矩阵 A 的行数和列数。
nrhs	host	input	矩阵 X 和 B 的列数。
A	device	input	<type> 数组,维度为 lda * n,其中 lda 不小于 max(1,n)。A 可以是下三角形的 Cholesky 因子 L，也可以是上三角形的 Cholesky 因子 U。
lda	host	input	用于存储矩阵 A 的二维数组前导维度。
devIpiv	device	input	大小至少为 n 的数组，包含主元索引。
B	device	in/out	大小为 ldb * nrhs 的 <type> 数组，其中 ldb 不小于 max(1,n)。作为输入，B 是右端量矩阵。作为输出，B 是解矩阵。
ldb	host	input	用于存储矩阵 B 的二维数组前导维度。
info	device	output	• 如果 info = 0，表示 Cholesky 分解成功。 • 如果 info = -i，表示第 i 个参数错误（不包含句柄）。 • 如果 info = i，则表示阶数为 i 的前导子矩阵不是正定的。

返回状态

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。
MCSOLVER_STATUS_NOT_INITIALIZED	库未进行初始化
MCSOLVER_STATUS_INVALID_VALUE	传递的参数无效 (n<0 或 lda<max(1,n))。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.2.9 mcsolverDnGetrs()

```
mcsolverStatus_t
mcsolverDnGetrs(
    mcsolverDnHandle_t handle,
    mcsolverDnParams_t params,
    mcblasOperation_t trans,
    int64_t n,
    int64_t nrhs,
    macaDataType dataTypeA,
    const void *A,
    int64_t lda,
    const int64_t *ipiv,
    macaDataType dataTypeB,
    void *B,
    int64_t ldb,
    int *info )
```

这个函数解决了一组包含多个右端量的线性方程组

$$op(A) * X = B$$

其中 A 是一个 n×n 矩阵，被 mcsolverDnGetrf LU 分解，即 A 的下三角部分是 L，A 的上三角部分 (包括对角线元素) 是 U。B 是一个使用通用 API 接口的 n×nrhs 右端矩阵。

输入参数 ipiv 是 mcsolverDnGetrf 的一个输出。它包含主指标，用于置换右端量。

如果输出参数 info = -i (小于 0), 那么第 i 个参数是错误的 (不包含句柄)。用户可以结合 mcsolverDnGetrf 和 mcsolverDnGetrs 来完成一个线性求解器。

目前，mcsolverDnGetrs 仅支持默认的算法

mcsolverDnGetrs 支持的算法表

mcsolver_ALG_0 or NULL	默认算法

mcsolverDnGetrs 的输入参数列表:

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库的上下文句柄。
params	host	input	由 mcsolverDnSetAdvOptions 收集的信息的结构体。
trans	host	input	操作 op(A) 非转置或（共轭）转置。
n	host	input	矩阵 A 的行数和列数。
nrhs	host	input	矩阵 X 和 B 的列数。
dataTypeA	host	in	数组 A 的数据类型
A	device	input	<type> 数组，维度为 lda * n，其中 lda 不小于 max(1,n)。A 可以是下三角形的 Cholesky 因子 L，也可以是上三角形的 Cholesky 因子 U。
lda	host	input	用于存储矩阵 A 的二维数组前导维度。
ipiv	device	input	长度至少为 n 的数组，包含主元索引。
dataTypeB	host	in	数组 B 的数据类型
B	device	in/out	<type> 数组，维度为 LDB * nrhs。LDB 不小于 max(1,n)。作为输入，B 是右边的矩阵。作为输出，B 是解矩阵。
ldb	host	input	用于存储矩阵 B 的二维数组前导维度。

下页继续

表 2.6 – 续上页

参数	内存	In/out	含义
info	device	output	• 如果 info = 0，表示 Cholesky 分解成功。 • 如果 info = -i，表示第 i 个参数有误 (不包含句柄)。 • 如果 info = i，则阶数为 i 的前导子矩阵不是正定的。

通用 API 有两种不同的类型，dataTypeA 是矩阵 A 的数据类型，dataTypeB 是矩阵 B 的数据类型。mcsolverDnGetrs 只支持以下四种组合。

数据类型和计算类型的有效组合

DataTypeA	dataTypeB	Meaning
MACA_R_32F	MACA_R_32F	SGETRS
MACA_R_64F	MACA_R_64F	DGETRS
MACA_C_32F	MACA_C_32F	CGETRS
MACA_C_64F	MACA_C_64F	ZGETRS

返回状态

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。
MCSOLVER_STATUS_NOT_INITIALIZED	库未进行初始化
MCSOLVER_STATUS_INVALID_VALUE	传递的参数无效 (n<0 或 lda<max(1,n))。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.2.10 mcsolverDn<t1><t2>gesv()

这些函数模仿了 LAPACK 中的 DSGESV 和 ZCGESV 函数。它们使用基于 LU 分解 Xgesv 的混合精度迭代细化技术计算带有一个或多个右端量的线性方程组的解。这些函数在功能方面类似于全精度 LU 求解器 (Xgesv，其中 X 表示 Z,C,D,S)，但它内部使用较低的精度，以提供更快的求解时间，由此得名混合精度。混合精度迭代细化技术意味着求解器以较低的精度计算 LU 分解，然后迭代细化这个解以达到输入/输出数据类型精度的精度。<t1> 对应输入/输出数据类型精度，<t2> 表示内部进行因式分解时的较低精度。

$$A \times X = B$$

其中 A 是 n-by-n 矩阵，X 和 B 是 n-by-nrhs 矩阵。

将函数 API 设计的尽可能接近 LAPACK API，可被视为快速、简单的替代方案。参数和行为与 LAPACK 的基本相同。下面将描述这些函数以及它们与 LAPACK 的区别。<t1><t2>gesv() 函数由两个浮点精度指定。<t1> 对应于主精度（例如，输入/输出数据类型精度），<t2> 表示内部进行因子分解时的较低精度。mcsolver<t1><t2>gesv() 首先尝试以较低的精度分解矩阵，并在迭代细化过程中使用这种分解，以获得与主精度 <t1> 相同的范数后向误差解 0，则该方法不能收敛，那么该方法将退回到主精度分解和求解 (Xgesv)，这样在这些函数的输出处总是有一个好的解。如果 <t2> 等于 <t1>，那么它不是一个混合精度过程，而是在相同的主精度下的一个全精度因子分解、求解和细化。

如果 ITER > ITERMAX，或者对于满足：RNRM < SQRT(N)*XNRM*ANRM*EPS*BWDMAX 的所有 RHS，迭代细化过程就会停止，其中：

- ITER 是迭代细化过程中当前迭代的次数
- RNRM 是残差的无穷范数
- XNRM 是解的无穷范数
- ANRM 是矩阵 A 的无穷算子范数
- EPS 是匹配 LAPACK <t1>LAMCH（‘epsilon’）的机器 epsilon

- ITERMAX 和 BWDMAX 的值分别固定为 50 和 1.0。

函数的返回值描述了求解的结果 mcsolver_STATUS_SUCCESS 表示函数成功结束，否则，它表示有一个 API 参数不正确，或者函数没有成功结束。关于错误的更多细节将在 `niters` 和 `dinfo` API 参数中。有关更多细节，请参见下面的描述。用户应该提供在设备内存上分配的所需工作区。所需的字节数可以通过调用相应的函数 `<t1><t2>gesv_bufferSize()` 来查询。

请注意，除了 LAPACK 中可用的两个混合精度函数（例如，`dsgesv` 和 `zcgesv`）之外，我们还提供了大量的混合精度函数，包括精度较低的 half、bfloat 和 tensorfloat 以及相同精度的函数 (例如，主精度和最低精度相等，<t2> 等于 <t1>)。下表指定了哪个接口函数将使用哪个精度。

当迭代细化求解器不能收敛到期望的精度（主精度，INOUT 数据精度）时，建议使用主精度作为内部最低精度，即 FP64 的 `mcsolverDn[DD,ZZ]gesv`。

支持 mcsolver <t1><t2>gesv() 的浮点精度组合

接口函数	主精度 (矩阵、rhs 和解的数据类型)	允许内部使用的最低精度
mcsolverDnZZgesv	mcDoubleComplex	double complex
mcsolverDnZCgesv	mcDoubleComplex	single complex
mcsolverDnZKgesv	mcDoubleComplex	half complex
mcsolverDnZEgesv	mcDoubleComplex	bfloat complex
mcsolverDnCCgesv	mcComplex	single complex
mcsolverDnCKgesv	mcComplex	half complex
mcsolverDnCEgesv	mcComplex	bfloat complex
mcsolverDnDDgesv	double	double
mcsolverDnDSgesv	double	single
mcsolverDnDHgesv	double	half
mcsolverDnDBgesv	double	bfloat
mcsolverDnSSgesv	float	single
mcsolverDnSHgesv	float	half
mcsolverDnSBgesv	float	bfloat

`mcsolverDn<t1><t2>gesv_bufferSize()` 函数将返回对应的 `mcsolverDn<t1><t2>gesv()` 函数所需的工作区缓冲区大小（以字节为单位）。

```
mcsolverStatus_t
mcsolverDnZZgesv_bufferSize(
    mcsolverHandle_t             handle,
    int                          n,
    int                          nrhs,
    mcDoubleComplex             *dA,
    int                          ldda,
    int                         *dipiv,
    mcDoubleComplex             *dB,
    int                          lddb,
    mcDoubleComplex             *dX,
    int                          lddx,
    void                        *dwork,
    size_t                      *lwork_bytes);

mcsolverStatus_t
mcsolverDnZCgesv_bufferSize(
    mcsolverHandle_t             handle,
    int                          n,
    int                          nrhs,
    mcDoubleComplex             *dA,
    int                          ldda,
    int                         *dipiv,
    mcDoubleComplex             *dB,
```

(下页继续)

(续上页)

```
    int                             lddb,
    mcDoubleComplex                *dX,
    int                             lddx,
    void                           *dwork,
    size_t                         *lwork_bytes);

mcsolverStatus_t
mcsolverDnZKgesv_bufferSize(
    mcsolverHandle_t                handle,
    int                             n,
    int                             nrhs,
    mcDoubleComplex                *dA,
    int                             ldda,
    int                            *dipiv,
    mcDoubleComplex                *dB,
    int                             lddb,
    mcDoubleComplex                *dX,
    int                             lddx,
    void                           *dwork,
    size_t                         *lwork_bytes);

mcsolverStatus_t
mcsolverDnZEgesv_bufferSize(
    mcsolverHandle_t                handle,
    int                             n,
    int                             nrhs,
    mcDoubleComplex                *dA,
    int                             ldda,
    int                            *dipiv,
    mcDoubleComplex                *dB,
    int                             lddb,
    mcDoubleComplex                *dX,
    int                             lddx,
    void                           *dwork,
    size_t                         *lwork_bytes);

mcsolverStatus_t
mcsolverDnZYgesv_bufferSize(
    mcsolverHandle_t                handle,
    int                             n,
    int                             nrhs,
    mcDoubleComplex                *dA,
    int                             ldda,
    int                            *dipiv,
    mcDoubleComplex                *dB,
    int                             lddb,
    mcDoubleComplex                *dX,
    int                             lddx,
    void                           *dwork,
    size_t                         *lwork_bytes);

mcsolverStatus_t
mcsolverDnCCgesv_bufferSize(
    mcsolverHandle_t                handle,
    int                             n,
    int                             nrhs,
```

(下页继续)

(续上页)

```
    mcComplex                   *dA,
    int                          ldda,
    int                         *dipiv,
    mcComplex                   *dB,
    int                          lddb,
    mcComplex                   *dX,
    int                          lddx,
    void                        *dwork,
    size_t                      *lwork_bytes);

mcsolverStatus_t
mcsolverDnCKgesv_bufferSize(
    mcsolverHandle_t             handle,
    int                          n,
    int                          nrhs,
    mcComplex                   *dA,
    int                          ldda,
    int                         *dipiv,
    mcComplex                   *dB,
    int                          lddb,
    mcComplex                   *dX,
    int                          lddx,
    void                        *dwork,
    size_t                      *lwork_bytes);

mcsolverStatus_t
mcsolverDnCEgesv_bufferSize(
    mcsolverHandle_t             handle,
    int                          n,
    int                          nrhs,
    mcComplex                   *dA,
    int                          ldda,
    int                         *dipiv,
    mcComplex                   *dB,
    int                          lddb,
    mcComplex                   *dX,
    int                          lddx,
    void                        *dwork,
    size_t                      *lwork_bytes);

mcsolverStatus_t
mcsolverDnCYgesv_bufferSize(
    mcsolverHandle_t             handle,
    int                          n,
    int                          nrhs,
    mcComplex                   *dA,
    int                          ldda,
    int                         *dipiv,
    mcComplex                   *dB,
    int                          lddb,
    mcComplex                   *dX,
    int                          lddx,
    void                        *dwork,
    size_t                      *lwork_bytes);

mcsolverStatus_t
```

(下页继续)

(续上页)

```
mcsolverDnDDgesv_bufferSize(
    mcsolverHandle_t             handle,
    int                          n,
    int                          nrhs,
    double                      *dA,
    int                          ldda,
    int                         *dipiv,
    double                      *dB,
    int                          lddb,
    double                      *dX,
    int                          lddx,
    void                        *dwork,
    size_t                      *lwork_bytes);

mcsolverStatus_t
mcsolverDnDSgesv_bufferSize(
    mcsolverHandle_t             handle,
    int                          n,
    int                          nrhs,
    double                      *dA,
    int                          ldda,
    int                         *dipiv,
    double                      *dB,
    int                          lddb,
    double                      *dX,
    int                          lddx,
    void                        *dwork,
    size_t                      *lwork_bytes);

mcsolverStatus_t
mcsolverDnDHgesv_bufferSize(
    mcsolverHandle_t             handle,
    int                          n,
    int                          nrhs,
    double                      *dA,
    int                          ldda,
    int                         *dipiv,
    double                      *dB,
    int                          lddb,
    double                      *dX,
    int                          lddx,
    void                        *dwork,
    size_t                      *lwork_bytes);

mcsolverStatus_t
mcsolverDnDBgesv_bufferSize(
    mcsolverHandle_t             handle,
    int                          n,
    int                          nrhs,
    double                      *dA,
    int                          ldda,
    int                         *dipiv,
    double                      *dB,
    int                          lddb,
    double                      *dX,
    int                          lddx,
```

(下页继续)

(续上页)

```
    void                        *dwork,
    size_t                      *lwork_bytes);

mcsolverStatus_t
mcsolverDnDXgesv_bufferSize(
    mcsolverHandle_t             handle,
    int                          n,
    int                          nrhs,
    double                      *dA,
    int                          ldda,
    int                         *dipiv,
    double                      *dB,
    int                          lddb,
    double                      *dX,
    int                          lddx,
    void                        *dwork,
    size_t                      *lwork_bytes);

mcsolverStatus_t
mcsolverDnSSgesv_bufferSize(
    mcsolverHandle_t             handle,
    int                          n,
    int                          nrhs,
    float                       *dA,
    int                          ldda,
    int                         *dipiv,
    float                       *dB,
    int                          lddb,
    float                       *dX,
    int                          lddx,
    void                        *dwork,
    size_t                      *lwork_bytes);

mcsolverStatus_t
mcsolverDnSHgesv_bufferSize(
    mcsolverHandle_t             handle,
    int                          n,
    int                          nrhs,
    float                       *dA,
    int                          ldda,
    int                         *dipiv,
    float                       *dB,
    int                          lddb,
    float                       *dX,
    int                          lddx,
    void                        *dwork,
    size_t                      *lwork_bytes);

mcsolverStatus_t
mcsolverDnSBgesv_bufferSize(
    mcsolverHandle_t             handle,
    int                          n,
    int                          nrhs,
    float                       *dA,
    int                          ldda,
    int                         *dipiv,
```

(下页继续)

(续上页)

```
    float                       *dB,
    int                          lddb,
    float                       *dX,
    int                          lddx,
    void                        *dwork,
    size_t                      *lwork_bytes);

mcsolverStatus_t
mcsolverDnSXgesv_bufferSize(
    mcsolverHandle_t             handle,
    int                          n,
    int                          nrhs,
    float                       *dA,
    int                          ldda,
    int                         *dipiv,
    float                       *dB,
    int                          lddb,
    float                       *dX,
    int                          lddx,
    void                        *dwork,
    size_t                      *lwork_bytes);
```

mcsolverDn<T1><T2>gesv_bufferSize() 函数的参数

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库上下文的句柄。
n	host	input	矩阵 A 的行数和列数。
nrhs	host	input	矩阵 X 和 B 的列数。
dA	device	None	大小为 n × n 的矩阵 A。可以是 NULL。
lda	host	input	用于存储矩阵 A 的二维数组前导维度。
dipiv	device	None	旋转序列。未使用，可以为 NULL。
dB	device	None	右端量 B 的集合，大小为 n × nrhs。可以是 NULL。
ipiv	device	input	长度至少为 n 的数组，包含主元索引。
lddb	host	input	用于存储右端矩阵 B 的二维数组的前导维数。ldb >= n。
dX	device	input	解向量的集合 X，大小为 n × nrhs。可以是 NULL。
lddx	host	input	用于存储解向量矩阵 X 的二维数组的前导维数。ldx >= n。
dwork	device	None	指向设备工作区的指针。未使用，可以为 NULL。
lwork_bytes	host	output	一个指向变量的指针，其中存储所需的临时工作区大小 (以字节为单位)。不能为 NULL。

```
mcsolverStatus_t mcsolverDnZZgesv(
        mcsolverDnHandle_t   handle,
        int                  n,
        int                  nrhs,
        mcDoubleComplex     *dA,
        int                  ldda,
        int                 *dipiv,
        mcDoubleComplex     *dB,
        int                  lddb,
        mcDoubleComplex     *dX,
        int                  lddx,
        void                *dWorkspace,
        size_t               lwork_bytes,
        int                 *niter,
        int                 *dinfo);
```

(下页继续)

(续上页)

```
mcsolverStatus_t mcsolverDnZCgesv(
        mcsolverDnHandle_t   handle,
        int                  n,
        int                  nrhs,
        mcDoubleComplex     *dA,
        int                  ldda,
        int                 *dipiv,
        mcDoubleComplex     *dB,
        int                  lddb,
        mcDoubleComplex     *dX,
        int                  lddx,
        void                *dWorkspace,
        size_t               lwork_bytes,
        int                 *niter,
        int                 *dinfo);

mcsolverStatus_t mcsolverDnZKgesv(
        mcsolverDnHandle_t   handle,
        int                  n,
        int                  nrhs,
        mcDoubleComplex     *dA,
        int                  ldda,
        int                 *dipiv,
        mcDoubleComplex     *dB,
        int                  lddb,
        mcDoubleComplex     *dX,
        int                  lddx,
        void                *dWorkspace,
        size_t               lwork_bytes,
        int                 *niter,
        int                 *dinfo);

mcsolverStatus_t mcsolverDnZEgesv(
        mcsolverDnHandle_t   handle,
        int                  n,
        int                  nrhs,
        mcDoubleComplex     *dA,
        int                  ldda,
        int                 *dipiv,
        mcDoubleComplex     *dB,
        int                  lddb,
        mcDoubleComplex     *dX,
        int                  lddx,
        void                *dWorkspace,
        size_t               lwork_bytes,
        int                 *niter,
        int                 *dinfo);

mcsolverStatus_t mcsolverDnZYgesv(
        mcsolverDnHandle_t   handle,
        int                  n,
        int                  nrhs,
        mcDoubleComplex     *dA,
        int                  ldda,
        int                 *dipiv,
```

(下页继续)

(续上页)

```
        mcDoubleComplex     *dB,
        int                  lddb,
        mcDoubleComplex     *dX,
        int                  lddx,
        void                *dWorkspace,
        size_t               lwork_bytes,
        int                 *niter,
        int                 *dinfo);

mcsolverStatus_t mcsolverDnCCgesv(
        mcsolverDnHandle_t   handle,
        int                  n,
        int                  nrhs,
        mcComplex           *dA,
        int                  ldda,
        int                 *dipiv,
        mcComplex           *dB,
        int                  lddb,
        mcComplex           *dX,
        int                  lddx,
        void                *dWorkspace,
        size_t               lwork_bytes,
        int                 *niter,
        int                 *dinfo);

mcsolverStatus_t mcsolverDnCKgesv(
        mcsolverDnHandle_t   handle,
        int                  n,
        int                  nrhs,
        mcComplex           *dA,
        int                  ldda,
        int                 *dipiv,
        mcComplex           *dB,
        int                  lddb,
        mcComplex           *dX,
        int                  lddx,
        void                *dWorkspace,
        size_t               lwork_bytes,
        int                 *niter,
        int                 *dinfo);

mcsolverStatus_t mcsolverDnCEgesv(
        mcsolverDnHandle_t   handle,
        int                  n,
        int                  nrhs,
        mcComplex           *dA,
        int                  ldda,
        int                 *dipiv,
        mcComplex           *dB,
        int                  lddb,
        mcComplex           *dX,
        int                  lddx,
        void                *dWorkspace,
        size_t               lwork_bytes,
        int                 *niter,
        int                 *dinfo);
```

(下页继续)

(续上页)

```
mcsolverStatus_t mcsolverDnCYgesv(
        mcsolverDnHandle_t   handle,
        int                  n,
        int                  nrhs,
        mcComplex           *dA,
        int                  ldda,
        int                 *dipiv,
        mcComplex           *dB,
        int                  lddb,
        mcComplex           *dX,
        int                  lddx,
        void                *dWorkspace,
        size_t               lwork_bytes,
        int                 *niter,
        int                 *dinfo);

mcsolverStatus_t mcsolverDnDDgesv(
        mcsolverDnHandle_t   handle,
        int                  n,
        int                  nrhs,
        double              *dA,
        int                  ldda,
        int                 *dipiv,
        double              *dB,
        int                  lddb,
        double              *dX,
        int                  lddx,
        void                *dWorkspace,
        size_t               lwork_bytes,
        int                 *niter,
        int                 *dinfo);

mcsolverStatus_t mcsolverDnDSgesv(
        mcsolverDnHandle_t   handle,
        int                  n,
        int                  nrhs,
        double              *dA,
        int                  ldda,
        int                 *dipiv,
        double              *dB,
        int                  lddb,
        double              *dX,
        int                  lddx,
        void                *dWorkspace,
        size_t               lwork_bytes,
        int                 *niter,
        int                 *dinfo);

mcsolverStatus_t mcsolverDnDHgesv(
        mcsolverDnHandle_t   handle,
        int                  n,
        int                  nrhs,
        double              *dA,
        int                  ldda,
        int                 *dipiv,
```

(下页继续)

(续上页)

```
        double              *dB,
        int                  lddb,
        double              *dX,
        int                  lddx,
        void                *dWorkspace,
        size_t               lwork_bytes,
        int                 *niter,
        int                 *dinfo);

mcsolverStatus_t mcsolverDnDBgesv(
        mcsolverDnHandle_t   handle,
        int                  n,
        int                  nrhs,
        double              *dA,
        int                  ldda,
        int                 *dipiv,
        double              *dB,
        int                  lddb,
        double              *dX,
        int                  lddx,
        void                *dWorkspace,
        size_t               lwork_bytes,
        int                 *niter,
        int                 *dinfo);

mcsolverStatus_t mcsolverDnDXgesv(
        mcsolverDnHandle_t   handle,
        int                  n,
        int                  nrhs,
        double              *dA,
        int                  ldda,
        int                 *dipiv,
        double              *dB,
        int                  lddb,
        double              *dX,
        int                  lddx,
        void                *dWorkspace,
        size_t               lwork_bytes,
        int                 *niter,
        int                 *dinfo);

mcsolverStatus_t mcsolverDnSSgesv(
        mcsolverDnHandle_t   handle,
        int                  n,
        int                  nrhs,
        float               *dA,
        int                  ldda,
        int                 *dipiv,
        float               *dB,
        int                  lddb,
        float               *dX,
        int                  lddx,
        void                *dWorkspace,
        size_t               lwork_bytes,
        int                 *niter,
        int                 *dinfo);
```

(下页继续)

(续上页)

```
mcsolverStatus_t mcsolverDnSHgesv(
        mcsolverDnHandle_t   handle,
        int                  n,
        int                  nrhs,
        float               *dA,
        int                  ldda,
        int                 *dipiv,
        float               *dB,
        int                  lddb,
        float               *dX,
        int                  lddx,
        void                *dWorkspace,
        size_t               lwork_bytes,
        int                 *niter,
        int                 *dinfo);

mcsolverStatus_t mcsolverDnSBgesv(
        mcsolverDnHandle_t   handle,
        int                  n,
        int                  nrhs,
        float               *dA,
        int                  ldda,
        int                 *dipiv,
        float               *dB,
        int                  lddb,
        float               *dX,
        int                  lddx,
        void                *dWorkspace,
        size_t               lwork_bytes,
        int                 *niter,
        int                 *dinfo);

mcsolverStatus_t mcsolverDnSXgesv(
        mcsolverDnHandle_t   handle,
        int                  n,
        int                  nrhs,
        float               *dA,
        int                  ldda,
        int                 *dipiv,
        float               *dB,
        int                  lddb,
        float               *dX,
        int                  lddx,
        void                *dWorkspace,
        size_t               lwork_bytes,
        int                 *niter,
        int                 *dinfo);
```

mcsolverDn<T1><T2>gesv() 函数的参数

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库上下文的句柄。
n	host	input	矩阵 A 的行数和列数。
nrhs	host	input	矩阵 X 和 B 的列数。

下页继续

表 2.8 – 续上页

参数	内存	In/out	含义
dA	device	None	大小为 n × n 的矩阵 A。可以为 NULL。
lda	host	input	用于存储矩阵 A 的二维数组前导维度。
dipiv	device	None	旋转序列。未使用，可以为 NULL。
dB	device	None	右端量 B 的集合，大小为 n × nrhs。可以是 NULL。
ipiv	device	input	长度至少为 n 的数组，包含主元索引。
lddb	host	input	用于存储右端矩阵 B 的二维数组的前导维数。ldb >= n。
dX	device	input	解向量的集合 X，大小为 n × nrhs。可以是 NULL。
lddx	host	input	用于存储解向量矩阵 X 的二维数组的前导维数。ldx >= n。
dwork	device	None	指向设备工作区的指针。未使用，可以为 NULL。
lwork_bytes	host	output	一个指向变量的指针，其中存储所需的临时工作区大小 (以字节为单位)。不能为 NULL。
niters	host	output	• 如果 iter < 0，迭代细化失败，主要精度 (Inputs/Outputs precision) 的因式分解已执行。 • 如果 iter = -1，考虑到机器参数、n、nrhs，从先验的角度上来说并不值得降低精度。 • 如果 iter = -2，当从主精度转换为低精度时会发生溢出。 • 如果 iter = -3，会在因式分解期间失败。 • 如果 iter = -5，会在计算过程中发生溢出。 • 如果 iter = -50，求解器在达到允许的最大迭代次数后，停止迭代细化。 • 如果 iter > 0，iter 是进行迭代细化的迭代次数。
dinfo	device	output	• 如果 info = 0，表示 Cholesky 分解成功。 • 如果 info = -i，表示第 i 个参数有误 (不包含句柄)。 • 如果 info = i，则阶数为 i 的前导子矩阵不是正定的。

返回状态

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。
MCSOLVER_STATUS_NOT_INITIALIZED	库未进行初始化。
MCSOLVER_STATUS_INVALID_VALUE	传递的参数无效 (n<0 或 lda<max(1,n))。
MCSOLVER_STATUS_IRS_OUT_OF_RANGE	与 niters <0 相关的数值误差，详情参见上表中 niters 的描述。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.2.11 mcsolverDn<t>geqrf()

这些辅助函数计算所需的工作缓冲区的大小。

```
mcsolverStatus_t
mcsolverDnSgeqrf_bufferSize(mcsolverDnHandle_t handle,
                            int m,
                            int n,
                            float *A,
```

(下页继续)

(续上页)

```
                        int lda,
                        int *Lwork );

mcsolverStatus_t
mcsolverDnDgeqrf_bufferSize(mcsolverDnHandle_t handle,
                        int m,
                        int n,
                        double *A,
                        int lda,
                        int *Lwork );

mcsolverStatus_t
mcsolverDnCgeqrf_bufferSize(mcsolverDnHandle_t handle,
                        int m,
                        int n,
                        mcComplex *A,
                        int lda,
                        int *Lwork );

mcsolverStatus_t
mcsolverDnZgeqrf_bufferSize(mcsolverDnHandle_t handle,
                        int m,
                        int n,
                        mcDoubleComplex *A,
                        int lda,
                        int *Lwork );
```

S 和 D 数据类型分别表示单精度实数和双精度实数。

```
mcsolverStatus_t
mcsolverDnSgeqrf(mcsolverDnHandle_t handle,
            int m,
            int n,
            float *A,
            int lda,
            float *TAU,
            float *Workspace,
            int Lwork,
            int *devInfo );

mcsolverStatus_t
mcsolverDnDgeqrf(mcsolverDnHandle_t handle,
            int m,
            int n,
            double *A,
            int lda,
            double *TAU,
            double *Workspace,
            int Lwork,
            int *devInfo );
```

C 和 Z 数据类型分别表示单精度复数和双精度复数。

```
mcsolverStatus_t
mcsolverDnCgeqrf(mcsolverDnHandle_t handle,
            int m,
            int n,
```

(下页继续)

(续上页)

```
            mcComplex *A,
            int lda,
            mcComplex *TAU,
            mcComplex *Workspace,
            int Lwork,
            int *devInfo );

mcsolverStatus_t
mcsolverDnZgeqrf(mcsolverDnHandle_t handle,
            int m,
            int n,
            mcDoubleComplex *A,
            int lda,
            mcDoubleComplex *TAU,
            mcDoubleComplex *Workspace,
            int Lwork,
            int *devInfo );
```

该函数计算 m×n 矩阵的 QR 分解

$$A = Q * R$$

其中 A 是一个 m×n 矩阵，Q 是一个 m×n 矩阵，R 是一个 n×n 上三角矩阵。

用户必须提供由输入参数 Workspace 指向的工作空间。输入参数 Lwork 表示工作空间的大小，由 geqrf_bufferSize() 返回。

矩阵 R 被覆盖到 A 的上三角部分，包括对角元素。

矩阵 Q 不是明确形成的，而是将一系列 householder 向量存储在 A 的下三角部分中。假设 householder 向量的前导非零元素为 1，这样输出参数 TAU 就包含了缩放因子 τ。如果 v 是原始的 householder 向量，则 q 是与 τ 相对应的新 householder 向量，满足以下关系式

$$I - 2 * v * v^H = I - tau * q * q^H$$

如果输出参数 devInfo = -i (小于 0)，则第 i 个参数是错误的 (不包含句柄)。

API of geqrf

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库的上下文句柄。
m	host	input	矩阵 A 的行数。
n	host	input	矩阵 A 的列数。
A	device	input	<type> 数组，维度为 lda * n，其中 lda 不小于 max(1,n)。A 可以是下三角形的 Cholesky 因子 L，也可以是上三角形的 Cholesky 因子 U。
lda	host	input	用于存储矩阵 A 的二维数组前导维度。
TAU	device	output	<type> 数组的维数至少为 min(m,n)。
Workspace	device	input	工作空间，<type> 大小为 Lwork 的数组。
info	device	output	• 如果 info = 0，则 Cholesky 分解成功。 • 如果 info = -i，表示第 i 个参数有误 (不包含句柄)。 • 如果 info = i，则阶数为 i 的前导子矩阵不是正定的。

返回状态

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。
MCSOLVER_STATUS_NOT_INITIALIZED	库未进行初始化
MCSOLVER_STATUS_INVALID_VALUE	传递的参数无效 (n<0 或 lda<max(1,n))。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.2.12 mcsolverDnGeqrf()

下面的辅助函数可以计算预分配缓冲区所需的大小。

```
mcsolverStatus_t
mcsolverDnGeqrf_bufferSize(
    mcsolverDnHandle_t handle,
    mcsolverDnParams_t params,
    int64_t m,
    int64_t n,
    macaDataType dataTypeA,
    const void *A,
    int64_t lda,
    macaDataType dataTypeTau,
    const void *tau,
    macaDataType computeType,
    size_t *workspaceInBytes )
```

例程如下：

```
mcsolverStatus_t
mcsolverDnGeqrf(
    mcsolverDnHandle_t handle,
    mcsolverDnParams_t params,
    int64_t m,
    int64_t n,
    macaDataType dataTypeA,
    void *A,
    int64_t lda,
    macaDataType dataTypeTau,
    void *tau,
    macaDataType computeType,
    void *pBuffer,
    size_t workspaceInBytes,
    int *info )
```

计算一个 m×n 矩阵的 QR 分解

$$A = Q * R$$

其中 A 是一个 m×n 矩阵，Q 是一个 m×n 矩阵，R 为使用通用 API 接口的 n×n 上三角矩阵。

用户必须提供由输入参数 pBuffer 指向的工作空间。输入参数 workspaceInBytes 是工作空间的字节大小，由 mcsolverDnGeqrf_bufferSize() 返回。

矩阵 R 被覆盖到 A 的上三角部分，包括对角元素。

矩阵 Q 不是明确形成的，而是将一系列 householder 向量存储在 A 的下三角部分。假设 householder 向量的前导非零元素为 1，这样输出参数 TAU 就包含了缩放因子 τ 0，则 v 是原始 householder 向量，则 q 是与 τ 相对应的新 householder 向量，满足以下关系式

$$I - 2 * v * v^H = I - \tau * q * q^H$$

如果输出参数 devInfo = -i （小于零），则第 i 个参数是错误的（不包含句柄）。

mcsolverDnGeqrf 支持的算法表

mcsolver_ALG_0 or NULL	默认算法

mcsolverDnGeqrf_bufferSize 和 mcsolverDnGeqrf 的输入参数列表：

API of geqrf

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库的上下文句柄。
params	host	input	由 mcsolverDnSetAdvOptions 收集的信息的结构体。
m	host	input	矩阵 A 的行数
n	host	input	矩阵 A 的列数
dataTypeA	host	in	数组 A 的数据类型
A	device	input	<type> 数组，维度为 lda * n，其中 lda 不小于 max(1,n)。A 可以是下三角形的 Cholesky 因子 L，也可以是上三角形的 Cholesky 因子 U。
lda	host	input	用于存储矩阵 A 的二维数组前导维度。
TAU	device	output	维数至少为 min(m,n) 的 <type> 数组。
computeType	host	in	计算的数据类型。
pBuffer	device	in/out	工作空间。大小为 workspaceInBytes 字节的 void 类型数组。
workspaceInBytes	host	input	工作数组 pBuffer 的大小（字节）。
info	device	output	• 如果 info = 0，Cholesky 分解成功。 • 如果 info = -i，表示第 i 个参数有误 (不包含句柄)。 • 如果 info = i，则阶数为 i 的前导子矩阵不是正定的。

通用 API 有两种不同的类型，dataTypeA 是矩阵 A 和数组 tau 的数据类型，computeType 是操作的计算类型。mcsolverDnGeqrf 只支持以下四种组合。

数据类型和计算类型的有效组合

DataTypeA	ComputeType	Meaning
MACA_R_32F	MACA_R_32F	SGEQRF
MACA_R_64F	MACA_R_64F	DGEQRF
MACA_C_32F	MACA_C_32F	CGEQRF
MACA_C_64F	MACA_C_64F	ZGEQRF

返回状态

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。
MCSOLVER_STATUS_NOT_INITIALIZED	库未进行初始化
MCSOLVER_STATUS_INVALID_VALUE	传递的参数无效 (n<0 或 lda<max(1,n))。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.2.13 mcsolverDn<t1><t2>gels()

这些函数使用基于 QR 分解 Xgels 的混合精度迭代细化技术计算具有一个或多个右侧的线性方程组的解。这些函数的功能与全精度 LAPACK QR（最小二乘）求解器（Xgels，其中 X 表示 Z、C、D、S）类似，但它在内部使用了较低的精度，以提供更快的求解时间，因此被称为混合精度。混合精度迭代细化技术是

指求解器以较低精度计算 QR 分解，然后迭代细化求解，以达到输入/输出数据类型精度。<t1> 对应于输入/输出数据类型精度，而 <t2> 表示内部较低的精度，分解将在该精度上进行。

```
mcsolverStatus_t mcsolverDnSBgels(
        mcsolverDnHandle_t      handle,
        int                     m,
        int                     n,
        int                     nrhs,
        float                *  dA,
        int                     ldda,
        float                *  dB,
        int                     lddb,
        float                *  dX,
        int                     lddx,
        void                 *  dWorkspace,
        size_t                  lwork_bytes,
        int                  *  niter,
        int                  *  dinfo);

mcsolverStatus_t mcsolverDnSXgels(
        mcsolverDnHandle_t      handle,
        int                     m,
        int                     n,
        int                     nrhs,
        float                *  dA,
        int                     ldda,
        float                *  dB,
        int                     lddb,
        float                *  dX,
        int                     lddx,
        void                 *  dWorkspace,
        size_t                  lwork_bytes,
        int                  *  niter,
        int                  *  dinfo);
```

mcsolverDn<T1><T2>gels() 函数的参数

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库的上下文句柄。
m	host	input	矩阵 A 的行数，应为非负数且 n <= m。
n	host	input	矩阵 A 的列数，应为非负数且 n <= m。
nrhs	host	input	要求解的右侧边数。应为非负数。
dA	device	in/out	矩阵 A，大小为 m-by-n，且不能为 NULL。返回时，如果最低精度不等于主要精度，且迭代细化求解器已收敛，则保持不变。否则，为垃圾值。
ldda	host	input	用于存储矩阵 A 的二维数组的前导维度。ldda >= m。
dB	device	input	右侧边集合 B，大小为 m-by-nrhs。且不能为 NULL。
lddb	host	input	用于存储右手边矩阵 B 的二维数组的前导维度。lddb >= max(1,m)。
dX	device	output	解向量矩阵集合 X，大小为 n-by-nrhs。不能为 NULL。
lddx	host	input	用于存储解向量矩阵集合 X 的二维数组前导维度。lddx >= max(1,n)。
dWorkspace	device	input	指向已分配设备内存中大小为 lwork_bytes 的工作空间。
lwork_bytes	host	input	工作空间的大小。至少应与由 mcsolverDn<T1><T2>gels_bufferSize() 函数返回的大小相同。

下页继续

表 2.10 – 续上页

参数	内存	In/out	含义
niters	host	output	• 如果 iter < 0，迭代细化失败，主要精度 (Inputs/Outputs precision) 的因式分解已执行。 • 如果 iter = -1，考虑到机器参数、n、nrhs，从先验的角度上来说并不值得降低精度。 • 如果 iter = -2，当从主精度转换为低精度时会发生溢出。 • 如果 iter = -3，会在因式分解期间失败。 • 如果 iter = -5，会在计算过程中发生溢出。 • 如果 iter = -50，求解器在达到允许的最大迭代次数后，停止迭代细化。 • 如果 iter > 0，iter 是进行迭代细化的迭代次数。
dinfo	device	output	返回的 IRS 求解器的状态值。如果为 0，则求解成功。如果 dinfo = i，则表示第 i 个参数无效。

返回状态

MCSOLVER_STATUS_SUCCESS	操作成功完成。
MCSOLVER_STATUS_NOT_INITIALIZED	库未初始化。
MCSOLVER_STATUS_INVALID_VALUE	传递了无效参数，比如： • n<0 • ldda<max(1,m) • lddb<max(1,m) • lddx<max(1,n)
MCSOLVER_STATUS_INVALID_WORKSPACE	lwork_bytes 小于所需的工作空间。
MCSOLVER_STATUS_IRS_OUT_OF_RANGE	与 niters<0 相关的数值误差，更多详情，参见上表中 niters 的描述。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败，更多详情，参见 dinfo 和 niters 参数。

2.4.2.14 mcsolverDn<t>ormqr()

这些辅助函数计算所需的工作缓冲区的大小。

```
mcsolverStatus_t
mcsolverDnSormqr_bufferSize(
    mcsolverDnHandle_t handle,
    mcblasSideMode_t side,
    mcblasOperation_t trans,
    int m,
    int n,
    int k,
    const float *A,
    int lda,
    const float *tau,
    const float *C,
    int ldc,
    int *lwork);

mcsolverStatus_t
mcsolverDnDormqr_bufferSize(
    mcsolverDnHandle_t handle,
```

(下页继续)

(续上页)

```
    mcblasSideMode_t side,
    mcblasOperation_t trans,
    int m,
    int n,
    int k,
    const double *A,
    int lda,
    const double *tau,
    const double *C,
    int ldc,
    int *lwork);

mcsolverStatus_t
mcsolverDnCunmqr_bufferSize(
    mcsolverDnHandle_t handle,
    mcblasSideMode_t side,
    mcblasOperation_t trans,
    int m,
    int n,
    int k,
    const mcComplex *A,
    int lda,
    const mcComplex *tau,
    const mcComplex *C,
    int ldc,
    int *lwork);

mcsolverStatus_t
mcsolverDnZunmqr_bufferSize(
    mcsolverDnHandle_t handle,
    mcblasSideMode_t side,
    mcblasOperation_t trans,
    int m,
    int n,
    int k,
    const mcDoubleComplex *A,
    int lda,
    const mcDoubleComplex *tau,
    const mcDoubleComplex *C,
    int ldc,
    int *lwork);
```

S 和 D 数据类型分别表示单精度实数和双精度实数。

```
mcsolverStatus_t
mcsolverDnSormqr(
    mcsolverDnHandle_t handle,
    mcblasSideMode_t side,
    mcblasOperation_t trans,
    int m,
    int n,
    int k,
    const float *A,
    int lda,
    const float *tau,
    float *C,
```

(下页继续)

(续上页)

```
    int ldc,
    float *work,
    int lwork,
    int *devInfo);

mcsolverStatus_t
mcsolverDnDormqr(
    mcsolverDnHandle_t handle,
    mcblasSideMode_t side,
    mcblasOperation_t trans,
    int m,
    int n,
    int k,
    const double *A,
    int lda,
    const double *tau,
    double *C,
    int ldc,
    double *work,
    int lwork,
    int *devInfo);
```

C 和 Z 数据类型分别表示单精度复数和双精度复数。

```
mcsolverStatus_t
mcsolverDnCunmqr(
    mcsolverDnHandle_t handle,
    mcblasSideMode_t side,
    mcblasOperation_t trans,
    int m,
    int n,
    int k,
    const mcComplex *A,
    int lda,
    const mcComplex *tau,
    mcComplex *C,
    int ldc,
    mcComplex *work,
    int lwork,
    int *devInfo);

mcsolverStatus_t
mcsolverDnZunmqr(
    mcsolverDnHandle_t handle,
    mcblasSideMode_t side,
    mcblasOperation_t trans,
    int m,
    int n,
    int k,
    const mcDoubleComplex *A,
    int lda,
    const mcDoubleComplex *tau,
    mcDoubleComplex *C,
    int ldc,
    mcDoubleComplex *work,
    int lwork,
    int *devInfo);
```

Q 是由 A 的 QR 分解（geqrf）中一系列初等反射向量组成的单元矩阵。

Q = H(1) H(2) ⋯H(k)

如果 side = MCBLAS_SIDE_LEFT，Q 是 m 阶；如果 side = MCBLAS_SIDE_RIGHT，则是 n 阶。

用户必须提供输入参数 work 所指向的工作空间。输入参数 lwork 表示工作空间的大小，由 geqrf_bufferSize() 或 ormqr_bufferSize() 返回。

如果输出参数 devInfo = -i （小于零），则 i-th 参数是错误的（不计句柄）。

用户可以结合 geqrf、ormqr 和 trsm 来完成线性求解器或最小平方求解器。

API of ormqr

参数	内存	输入/输出	含义
handle	host	input	指向 mcsolverDn 库上下文的句柄。
side	host	input	表示矩阵 Q 是在 C 的左边还是右边。
trans	host	input	非转置或（共轭）转置的操作 op(Q)。
m	host	input	矩阵 C 的行数。
n	host	input	矩阵 C 的列数。
k	host	input	其乘积定义矩阵 Q 的初等反射数量。
A	device	in/out	<type> 维数为 lda * k 的数组，其中 lda 不小于 max(1, m)。矩阵 A 来自 geqrf，因此第 i 列包含初等反射向量。
lda	host	input	用于存储矩阵 A 的二维数组的前导维数。如果 side 是 MCBLAS_SIDE_LEFT，lda >= max(1,m)；如果 side 是 MCBLAS_SIDE_RIGHT，lda >= max(1,n)
tau	device	output	<type> 数组，维数至少为 min(m,n)。向量 tau 来自 geqrf，因此 tau(i) 是第 i 个初等反射向量的标量。
C	device	in/out	<type> 大小为 ldc * n 的数组。退出时，C 被 op(Q)*C 重写。
ldc	host	input	矩阵 C 的二维数组的前导维数。ldc >= max(1,m)。
work	device	in/out	工作空间，大小为 lwork 的 <type> 数组。
lwork	host	input	工作数组 work 的大小。
devInfo	device	output	如果 devInfo = 0，ormqr 成功。如果 devInfo = -i，说明 i-th 参数有误（不包括句柄）。

返回状态

MCSOLVER_STATUS_SUCCESS	操作成功完成。
MCSOLVER_STATUS_NOT_INITIALIZED	库未初始化。
MCSOLVER_STATUS_INVALID_VALUE	传递了无效参数（m,n<0 或错误的 lda 或 ldc）。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.2.15 mcsolverDn<t>orgqr()

这些辅助函数计算所需的工作缓冲区的大小。

```
mcsolverStatus_t
mcsolverDnSorgqr_bufferSize(
    mcsolverDnHandle_t handle,
    int m,
    int n,
    int k,
    const float *A,
    int lda,
    const float *tau,
```

(下页继续)

(续上页)

```
    int *lwork);

mcsolverStatus_t
mcsolverDnDorgqr_bufferSize(
    mcsolverDnHandle_t handle,
    int m,
    int n,
    int k,
    const double *A,
    int lda,
    const double *tau,
    int *lwork);

mcsolverStatus_t
mcsolverDnCungqr_bufferSize(
    mcsolverDnHandle_t handle,
    int m,
    int n,
    int k,
    const mcComplex *A,
    int lda,
    const mcComplex *tau,
    int *lwork);

mcsolverStatus_t
mcsolverDnZungqr_bufferSize(
    mcsolverDnHandle_t handle,
    int m,
    int n,
    int k,
    const mcDoubleComplex *A,
    int lda,
    const mcDoubleComplex *tau,
    int *lwork);
```

S 和 D 数据类型分别表示单精度实数和双精度实数。

```
mcsolverStatus_t
mcsolverDnSorgqr(
    mcsolverDnHandle_t handle,
    int m,
    int n,
    int k,
    float *A,
    int lda,
    const float *tau,
    float *work,
    int lwork,
    int *devInfo);

mcsolverStatus_t
mcsolverDnDorgqr(
    mcsolverDnHandle_t handle,
    int m,
    int n,
    int k,
```

(下页继续)

(续上页)

```
    double *A,
    int lda,
    const double *tau,
    double *work,
    int lwork,
    int *devInfo);
```

C 和 Z 数据类型分别表示单精度复数和双精度复数。

```
mcsolverStatus_t
mcsolverDnCungqr(
    mcsolverDnHandle_t handle,
    int m,
    int n,
    int k,
    mcComplex *A,
    int lda,
    const mcComplex *tau,
    mcComplex *work,
    int lwork,
    int *devInfo);

mcsolverStatus_t
mcsolverDnZungqr(
    mcsolverDnHandle_t handle,
    int m,
    int n,
    int k,
    mcDoubleComplex *A,
    int lda,
    const mcDoubleComplex *tau,
    mcDoubleComplex *work,
    int lwork,
    int *devInfo);
```

该函数用以下方式覆盖 m×n 矩阵 A

$$Q = H(1) * H(2) * ... * H(k)$$

其中 Q 是由存储在 A 中的一系列初等反射向量组成的酉矩阵。

MCSOLVER_STATUS_SUCCESS	操作成功完成。
MCSOLVER_STATUS_NOT_INITIALIZED	库未初始化。
MCSOLVER_STATUS_INVALID_VALUE	传递了无效参数（m,n,k<0,n>m,k>n 或者 lda<m）。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.2.16 mcsolverDn<t>sytrf()

这些辅助函数会计算所需缓冲区的大小。

```
mcsolverStatus_t
mcsolverDnSsytrf_bufferSize(mcsolverDnHandle_t handle,
                            int n,
                            float *A,
                            int lda,
```

(下页继续)

(续上页)

```
                        int *Lwork );

mcsolverStatus_t
mcsolverDnDsytrf_bufferSize(mcsolverDnHandle_t handle,
                        int n,
                        double *A,
                        int lda,
                        int *Lwork );

mcsolverStatus_t
mcsolverDnCsytrf_bufferSize(mcsolverDnHandle_t handle,
                        int n,
                        mcComplex *A,
                        int lda,
                        int *Lwork );

mcsolverStatus_t
mcsolverDnZsytrf_bufferSize(mcsolverDnHandle_t handle,
                        int n,
                        mcDoubleComplex *A,
                        int lda,
                        int *Lwork );
```

S 和 D 数据类型分别表示单精度实数和双精度实数。

```
mcsolverStatus_t
mcsolverDnSsytrf(mcsolverDnHandle_t handle,
            mcblasFillMode_t uplo,
            int n,
            float *A,
            int lda,
            int *ipiv,
            float *work,
            int lwork,
            int *devInfo );

mcsolverStatus_t
mcsolverDnDsytrf(mcsolverDnHandle_t handle,
            mcblasFillMode_t uplo,
            int n,
            double *A,
            int lda,
            int *ipiv,
            double *work,
            int lwork,
            int *devInfo );
```

C 和 Z 数据类型分别表示单精度复数和双精度复数。

```
mcsolverStatus_t
mcsolverDnCsytrf(mcsolverDnHandle_t handle,
            mcblasFillMode_t uplo,
            int n,
            mcComplex *A,
            int lda,
            int *ipiv,
            mcComplex *work,
```

(下页继续)

(续上页)

```
            int lwork,
            int *devInfo );

mcsolverStatus_t
mcsolverDnZsytrf(mcsolverDnHandle_t handle,
            mcblasFillMode_t uplo,
            int n,
            mcDoubleComplex *A,
            int lda,
            int *ipiv,
            mcDoubleComplex *work,
            int lwork,
            int *devInfo );
```

此函数用于计算一个 n×n 对称不定矩阵的 Bunch-Kaufman 分解。A 是一个 n×n 对称矩阵，只有上三角或下三角有意义。输入参数 uplo 在矩阵的某个部分被使用。函数将保留其他部分不变。

如果输入参数 uplo 为 MCBLAS_FILL_MODE_LOWER，则只处理 A 的下三角部分，并用下三角因子 L 和分块对角矩阵 D 代替。每块 D 都是 1x1 或 2x2 块，取决于主元位置

$$P * A * P^T = L * D * L^T$$

如果输入参数 uplo 为 MCBLAS_FILL_MODE_UPPER，仅处理 A 的上三角形部分，并替换为上三角因子 U 和对角分块矩阵 D。

$$P * A * P^T = U * D * U^T$$

用户必须提供输入参数 work 所指向的工作空间。输入参数 lwork 表示工作空间的大小，由 sytrf_bufferSize() 返回。如果 Bunch-Kaufman 分解失败，即 A 是单数。输出参数 devInfo`i 将表示为 D(i,i)=0。

如果输出参数 devInfo = -i （小于零），则第 i 个参数是错误的（不包含句柄）。

输出参数 devIpiv 包含主元序列。如果 devIpiv(i) = k >= 0，D(i,i) 是 1x1 块，则 A 的第 i 个行/列与 A 的 k-th 行/列互换。如果 uplo 为 MCBLAS_FILL_MODE_UPPER 且 devIpiv(i-1) = devIpiv(i) = -m < 0，D(i-1:i,i-1:i) 为 2x2 块，则 (i-1)-th 行/列与 m-th 行/列互换。如果 uplo 是 MCBLAS_FILL_MODE_LOWER 且 devIpiv(i+1) = devIpiv(i) = -m < 0，D(i:i+1,i:i+1) 是一个 2x2 块，则 (i+1)-th 行/列与 m-th 行/列互换。

API of sytrf

参数	内存	In/out	含义
handle	host	input	指向 mcsolverDN 库上下文的句柄。
uplo	host	input	表示矩阵 A 的下三角或上三角部分是否存储，且另一部分不被引用。
n	host	input	矩阵 A 的行数和列数。
A	device	in/out	<type> 维度为 lda * n 的数组，其中 lda 不小于 max(1,n)。
lda	host	input	用于存储矩阵 A 的二维数组的前导维度。
ipiv	device	output	大小至少为 n 的数组，包含主元索引。
work	device	in/out	工作空间，<type> 大小为 lwork 的数组。
lwork	host	input	工作空间 work 的大小。
devInfo	device	output	如果 devInfo = 0，LU 分解成功。如果 devInfo = -i，第 i 个参数有误（不包括句柄）。如果 devInfo = i，则 D(i,i) = 0。

Status Returned

返回状态

MCSOLVER_STATUS_SUCCESS	操作成功完成。
MCSOLVER_STATUS_NOT_INITIALIZED	库未初始化。
MCSOLVER_STATUS_INVALID_VALUE	传递了无效参数 (n<0 或 lda<max(1,n))。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.2.17 mcsolverDn<t>potrfBatched()

S 和 D 数据类型分别为单精度实数和双精度实数。

```
mcsolverStatus_t
mcsolverDnSpotrfBatched(
    mcsolverDnHandle_t handle,
    mcblasFillMode_t uplo,
    int n,
    float *Aarray[],
    int lda,
    int *infoArray,
    int batchSize);

mcsolverStatus_t
mcsolverDnDpotrfBatched(
    mcsolverDnHandle_t handle,
    mcblasFillMode_t uplo,
    int n,
    double *Aarray[],
    int lda,
    int *infoArray,
    int batchSize);
```

C 和 Z 数据类型分别为单精度复数和双精度复数。

```
mcsolverStatus_t
mcsolverDnCpotrfBatched(
    mcsolverDnHandle_t handle,
    mcblasFillMode_t uplo,
    int n,
    mcComplex *Aarray[],
    int lda,
    int *infoArray,
    int batchSize);

mcsolverStatus_t
mcsolverDnZpotrfBatched(
    mcsolverDnHandle_t handle,
    mcblasFillMode_t uplo,
    int n,
    mcDoubleComplex *Aarray[],
    int lda,
    int *infoArray,
    int batchSize);
```

该函数计算一系列 Hermitian 正定矩阵的 Cholesky 分解。

对于每个 i=0,1,...,batchSize-1，Aarray[i] 是一个 n×n 的 Hermitian 矩阵，只有下三角部分或上三角部分是有意义的。输入参数 uplo 指示了矩阵的哪一部分被使用。

如果输入参数 uplo 是 MCBLAS_FILL_MODE_UPPER，则只处理矩阵 A 的下三角部分，并用下三角

Cholesky 因子 L 替换它。

$$A = L * L^H$$

如果输入参数 uplo 是 MCBLAS_FILL_MODE_UPPER，则只处理矩阵 A 的上三角部分，并用上三角 Cholesky 因子 U 替换它。

$$A = U^H * U$$

如果 Cholesky 分解失败，即矩阵 A 的某个前导子阵不是正定的，或 L 或 U 的某些对角元素不是实数。输出参数 infoArray 将指示在矩阵 A 中不是正定的最小前导子阵。

infoArray 是一个大小为 batchsize 的整数数组。如果 potrfBatched 返回 MCSOLVER_STATUS_INVALID_VALUE，则 infoArray[0] = -i（小于零），这表示第 i 个参数错误（不包括句柄）。如果 potrfBatched 返回 MCSOLVER_STATUS_SUCCESS 但 infoArray[i] = k 是正数，则第 i 个矩阵不是正定的，并且 Cholesky 分解在第 k 行失败。

备注：矩阵 A 的另一部分用作工作空间。例如，如果 uplo 是 MCBLAS_FILL_MODE_UPPER，则矩阵 A 的上三角部分包含 Cholesky 因子 U，并且在 potrfBatched 后，矩阵 A 的下三角部分将被破坏。

API of potrfBatched

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库上下文句柄。
uplo	host	input	表示下三角或上三角部分是否被存储；另一部分被用作工作空间。
n	host	input	矩阵 A 的行数和列数。
Aarray	device	in/out	大小为 lda * n <type> 数组的指针数组，其中 lda 至少不小于 max(1,n)。
lda	host	input	二维数组的前导维度，用于存储每个矩阵 Aarray[i]。
infoArray	device	output	大小为 batchSize 的数组。infoArray[i] 包含对 Aarray[i] 的分解信息。如果 potrfBatched 返回 MCSOLVER_STATUS_INVALID_VALUE，则 infoArray[0] = -i（小于零）表示第 i 个参数错误（不包括句柄）。如果 potrfBatched 返回 MCSOLVER_STATUS_SUCCESS，则 infoArray[i] = 0 表示第 i 个矩阵的 Cholesky 分解成功，而 infoArray[i] = k 表示第 i 个矩阵的大小为 k 的前导子矩阵不是正定的。
batchSize	host	input	Aarray 中的指针数量。

返回状态

MCSOLVER_STATUS_SUCCESS	操作成功完成。
MCSOLVER_STATUS_NOT_INITIALIZED	该库尚未初始化。
MCSOLVER_STATUS_INVALID_VALUE	传递了无效参数（n<0 或 lda<max(1,n) 或 batchSize<1）。
MCSOLVER_STATUS_INTERNAL_ERROR	一个内部操作失败。

2.4.2.18 mcsolverDn<t>potrsBatched()

```
mcsolverStatus_t
mcsolverDnSpotrsBatched(
    mcsolverDnHandle_t handle,
    mcblasFillMode_t uplo,
    int n,
    int nrhs,
```

(下页继续)

(续上页)

```
    float *Aarray[],
    int lda,
    float *Barray[],
    int ldb,
    int *info,
    int batchSize);

mcsolverStatus_t
mcsolverDnDpotrsBatched(
    mcsolverDnHandle_t handle,
    mcblasFillMode_t uplo,
    int n,
    int nrhs,
    double *Aarray[],
    int lda,
    double *Barray[],
    int ldb,
    int *info,
    int batchSize);

mcsolverStatus_t
mcsolverDnCpotrsBatched(
    mcsolverDnHandle_t handle,
    mcblasFillMode_t uplo,
    int n,
    int nrhs,
    mcComplex *Aarray[],
    int lda,
    mcComplex *Barray[],
    int ldb,
    int *info,
    int batchSize);

mcsolverStatus_t
mcsolverDnZpotrsBatched(
    mcsolverDnHandle_t handle,
    mcblasFillMode_t uplo,
    int n,
    int nrhs,
    mcDoubleComplex *Aarray[],
    int lda,
    mcDoubleComplex *Barray[],
    int ldb,
    int *info,
    int batchSize);
```

此函数解决一系列线性系统

$$A[i] * X[i] = B[i]$$

其中每个 `Aarray[i] for i=0,1,...,batchSize-1` 都是一个 `n×n` 的 Hermitian 矩阵，只有下三角或上三角部分是有意义的。输入参数 `uplo` 表示矩阵的哪个部分被使用。

用户首先必须调用 `potrfBatched` 来因子分解矩阵 `Aarray[i]` 。如果输入参数 `uplo` 为 `MCBLAS_FILL_MODE_LOWER` ，则矩阵 `A` 与下三角的 Cholesky 分解因子 `L` 对应于:math:A = L*L^{H}。如果输入参数 `uplo` 为 `MCBLAS_FILL_MODE_UPPER` ，则矩阵 `A` 与上三角的 Cholesky 分解因子 `U` 对应于： $A = U^H * U$ 。

此操作是原地操作，即矩阵 `X` 覆盖了具有相同领先维度 `ldb` 的矩阵 `B` 。

输出参数 info 是一个标量。如果 info = -i （小于零），则第 i 个参数错误（不包括句柄）。

备注 1：仅支持 nrhs=1 。

备注 2：来自 potrfBatched 的 infoArray 表示矩阵是否正定。来自 potrsBatched 的 info 仅显示哪个输入参数错误（不包括句柄）。

备注 3：A 的另一部分用作工作空间。例如，如果 uplo 为 MCBLAS_FILL_MODE_UPPER ，则矩阵 A 的上三角包含 Cholesky 分解因子 U ，而下三角部分在 potrsBatched 之后被销毁。

API of potrsBatched

参数	内存	In/out	含义
handle	host	input	指向 mcsolveDN 库上下文的句柄。
uplo	host	input	指示矩阵 A 的下三角部分或上三角部分是否存储。
n	host	input	矩阵 A 的行数和列数。
nrhs	host	input	矩阵 X 和 B 的列数。
Aarray	device	in/out	指向 <type> 数组的指针，维数为 lda * n ，其中 lda 不小于 max(1,n) 。Aarray[i] 是下三角 Cholesky 因子 L 或上三角 Cholesky 因子 U 。
lda	host	input	用于存储每个矩阵 Aarray[i] 的二维数组的前导维度。
Barray	device	in/out	指向 <type> 数组的指针，维数为 ldb * nrhs 。ldb 不小于 max(1,n) 。作为输入，Barray[i] 是右侧的矩阵。作为输出，Barray[i] 是解矩阵。
ldb	host	input	用于存储每个矩阵 Barray[i] 的二维数组的前导维度。
info	device	output	如果 info = 0 ，则所有参数正确。如果 info = -i ，则第 i 个参数不正确（不包括句柄）。
batchSize	host	input	Aarray 中的指针数。

返回状态

MCSOLVER_STATUS_SUCCESS	操作成功完成。
MCSOLVER_STATUS_NOT_INITIALIZED	库未初始化。
MCSOLVER_STATUS_INVALID_VALUE	传递了无效的参数 (n<0 、nrhs<0 、lda<max(1,n) 、ldb<max(1,n) 或 batchSize<0)。
MCSOLVER_STATUS_INTERNAL_ERROR	一个内部操作失败。

2.4.3 Dense 特征值求解器参考 (legacy)

这一章节描述了 mcsolverDN 的特征值求解器 API，包括双对角化和奇异值分解（SVD）功能。

2.4.3.1 mcsolverDn<t>gebrd()

这些辅助函数计算所需的工作缓冲区的大小。

```
mcsolverStatus_t
mcsolverDnSgebrd_bufferSize(
    mcsolverDnHandle_t handle,
    int m,
    int n,
    int *Lwork );

mcsolverStatus_t
mcsolverDnDgebrd_bufferSize(
    mcsolverDnHandle_t handle,
```

(下页继续)

(续上页)

```
    int m,
    int n,
    int *Lwork );

mcsolverStatus_t
mcsolverDnCgebrd_bufferSize(
    mcsolverDnHandle_t handle,
    int m,
    int n,
    int *Lwork );

mcsolverStatus_t
mcsolverDnZgebrd_bufferSize(
    mcsolverDnHandle_t handle,
    int m,
    int n,
    int *Lwork );
```

S 和 D 数据类型分别表示单精度实数和双精度实数。

```
mcsolverStatus_t
mcsolverDnSgebrd(mcsolverDnHandle_t handle,
            int m,
            int n,
            float *A,
            int lda,
            float *D,
            float *E,
            float *TAUQ,
            float *TAUP,
            float *Work,
            int Lwork,
            int *devInfo );

mcsolverStatus_t
mcsolverDnDgebrd(mcsolverDnHandle_t handle,
            int m,
            int n,
            double *A,
            int lda,
            double *D,
            double *E,
            double *TAUQ,
            double *TAUP,
            double *Work,
            int Lwork,
            int *devInfo );
```

C 和 Z 数据类型分别表示单精度复数和双精度复数。

```
mcsolverStatus_t
mcsolverDnCgebrd(mcsolverDnHandle_t handle,
            int m,
            int n,
            mcComplex *A,
            int lda,
            float *D,
```

(下页继续)

(续上页)

```
            float *E,
            mcComplex *TAUQ,
            mcComplex *TAUP,
            mcComplex *Work,
            int Lwork,
            int *devInfo );

mcsolverStatus_t
mcsolverDnZgebrd(mcsolverDnHandle_t handle,
            int m,
            int n,
            mcDoubleComplex *A,
            int lda,
            double *D,
            double *E,
            mcDoubleComplex *TAUQ,
            mcDoubleComplex *TAUP,
            mcDoubleComplex *Work,
            int Lwork,
            int *devInfo );
```

这个函数通过正交变换 $Q^H * A * P = B$ 将一个一般的 m×n 矩阵 A 简化为一个实数上三角或下三角的双对角形式矩阵 B。如果 m>=n，则 B 是上三角双对角形式；如果 m<n，则 B 是下三角双对角形式。

矩阵 Q 和 P 以如下方式覆盖到矩阵 A 中：

- 如果 m>=n，则对角线和第一个超对角线会被上三角双对角矩阵 B 覆盖。在对角线以下的元素，通过数组 TAUQ 表示正交矩阵 Q 为一系列初等反射的乘积；在第一个超对角线以上的元素，通过数组 TAUP 表示正交矩阵 P 为一系列初等反射的乘积。
- 如果 m<n，则对角线和第一个次对角线会被下三角双对角矩阵 B 覆盖。在第一个次对角线以下的元素，通过数组 TAUQ 表示正交矩阵 Q 为一系列初等反射的乘积；在对角线以上的元素，通过数组 TAUP 表示正交矩阵 P 为一系列初等反射的乘积。

用户必须提供输入参数 Work 所指向的工作空间。输入参数 Lwork 表示工作空间的大小，可以通过 gebrd_bufferSize() 函数返回。

如果输出参数 devInfo = -i (小于 0)，第 i 个参数是错误的 (不包含句柄)。

备注：gebrd 仅支持 m>=n。

API of gebrd

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库上下文的句柄。
m	host	input	矩阵 A 的行数。
n	host	input	矩阵 A 的列数。
A	device	in/out	<type> 维度为 lda * n 的数组，其中 lda 不小于 max(1,n)。
lda	host	input	用于存储矩阵 A 的二维数组前导维度。
D	device	output	维度为 min(m,n) 的实数数组。分解后的双对角矩阵 B 的对角元素：D(i) = A(i,i)。
E	device	output	维度为 min(m,n) 的实数数组。分解后的双对角矩阵 B 的非对角元素：若 m>=n，则对于 i = 1,2,...,n-1，有 E(i) = A(i,i+1)；若 m<n，则对于 i = 1,2,...,m-1，有 E(i) = A(i+1,i)。
TAUQ	device	output	<type> 维度为 min(m,n) 的数组。表示正交矩阵 Q 的基本反射器的标量因子。
TAUP	device	output	<type> 维度为 min(m,n) 的数组。表示正交矩阵 P 的基本反射器的标量因子。

下页继续

表 2.15 – 续上页

参数	内存	In/out	含义
Work	device	in/out	工作空间，<type> 大小为 Lwork 的数组。
Lwork	host	input	由 gebrd_bufferSize 返回的 Work 大小。
devInfo	device	output	若 devInfo = 0，则操作成功。若 devInfo = -i，则第 i 个参数错误（不计入句柄）。

返回状态

MCSOLVER_STATUS_SUCCESS	操作成功完成。
MCSOLVER_STATUS_NOT_INITIALIZED	库未初始化。
MCSOLVER_STATUS_INVALID_VALUE	传递了无效参数 (m,n<0 或 lda<max(1,m))。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.3.2 mcsolverDn<t>orgbr()

这些辅助函数计算所需的工作缓冲区的大小。

```
mcsolverStatus_t
mcsolverDnSorgbr_bufferSize(
    mcsolverDnHandle_t handle,
    mcblasSideMode_t side,
    int m,
    int n,
    int k,
    const float *A,
    int lda,
    const float *tau,
    int *lwork);

mcsolverStatus_t
mcsolverDnDorgbr_bufferSize(
    mcsolverDnHandle_t handle,
    mcblasSideMode_t side,
    int m,
    int n,
    int k,
    const double *A,
    int lda,
    const double *tau,
    int *lwork);

mcsolverStatus_t
mcsolverDnCungbr_bufferSize(
    mcsolverDnHandle_t handle,
    mcblasSideMode_t side,
    int m,
    int n,
    int k,
    const mcComplex *A,
    int lda,
    const mcComplex *tau,
    int *lwork);

mcsolverStatus_t
```

(下页继续)

(续上页)

```
mcsolverDnZungbr_bufferSize(
    mcsolverDnHandle_t handle,
    mcblasSideMode_t side,
    int m,
    int n,
    int k,
    const mcDoubleComplex *A,
    int lda,
    const mcDoubleComplex *tau,
    int *lwork);
```

S 和 D 数据类型分别表示单精度实数和双精度实数。

```
mcsolverStatus_t
mcsolverDnSorgbr(
    mcsolverDnHandle_t handle,
    mcblasSideMode_t side,
    int m,
    int n,
    int k,
    float *A,
    int lda,
    const float *tau,
    float *work,
    int lwork,
    int *devInfo);

mcsolverStatus_t
mcsolverDnDorgbr(
    mcsolverDnHandle_t handle,
    mcblasSideMode_t side,
    int m,
    int n,
    int k,
    double *A,
    int lda,
    const double *tau,
    double *work,
    int lwork,
    int *devInfo);
```

C 和 Z 数据类型分别表示单精度复数和双精度复数。

```
mcsolverStatus_t
mcsolverDnCungbr(
    mcsolverDnHandle_t handle,
    mcblasSideMode_t side,
    int m,
    int n,
    int k,
    mcComplex *A,
    int lda,
    const mcComplex *tau,
    mcComplex *work,
    int lwork,
    int *devInfo);
```

(下页继续)

(续上页)

```
mcsolverStatus_t
mcsolverDnZungbr(
    mcsolverDnHandle_t handle,
    mcblasSideMode_t side,
    int m,
    int n,
    int k,
    mcDoubleComplex *A,
    int lda,
    const mcDoubleComplex *tau,
    mcDoubleComplex *work,
    int lwork,
    int *devInfo);
```

这个函数在将矩阵 A 简化为双对角形式时，生成由 gebrd 确定的酉矩阵 Q 或 P**H ，简化公式为 $Q^H * A * P = B$。Q 和 P**H 分别被定义为初等反射变换 H(i) 或 G(i) 的乘积。

用户必须提供输入参数 work 所指向的工作空间。输入参数 lwork 表示工作空间的大小，可以通过 orgbr_bufferSize() 函数返回。

如果输出参数 devInfo = -i (小于 0)，则表示第 i 个参数是错误的 (不包含句柄)。

API of orgbr

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库上下文的句柄。
side	host	input	若 side = MCBLAS_SIDE_LEFT ，则生成 Q 。若 side = MCBLAS_SIDE_RIGHT，则生成 P**T。
m	host	input	矩阵 Q 或 P**T 的行数。
n	host	input	若 side = MCBLAS_SIDE_LEFT ，则 m >= n >= min(m, k) 。若 side = MCBLAS_SIDE_RIGHT ，则 n >= m >= min(n, k) 。
k	host	input	若 side = MCBLAS_SIDE_LEFT ，则为被 gebrd 缩减的原始 m-by-k 矩阵的列数。若 side = MCBLAS_SIDE_RIGHT ，则为被 gebrd 缩减的原始 k-by-n 矩阵的行数。
A	device	in/out	<type> 维度为 lda * n 的数组。在输入时，这些向量定义了由 gebrd 返回的基本反射器。在输出时，为 m-by-n 矩阵 Q 或 P**T 。
lda	host	input	用于存储矩阵 A 的二维数组前导维度。lda >= max(1, m)。
tau	device	output	<type> 维度为 min(m, k) （若 side 为 MCBLAS_SIDE_LEFT）；维度为 min(n, k) （若 side 为 MCBLAS_SIDE_RIGHT ）的数组。tau(i) 必须包含由 gebrd 在其数组参数 TAUQ 或 TAUP 中返回的基本反射器 H(i) 或 G(i) 标量因子，用以决定 Q 或 P**T 。
work	device	in/out	工作空间，<type> 大小为 lwork 的数组。
lwork	host	input	工作数组 work 的大小。
devInfo	device	output	若 info = 0 ，则 ormqr 操作成功。若 info = -i ，则第 i 个参数错误（不计入句柄）。

返回状态

MCSOLVER_STATUS_SUCCESS	操作成功完成。
MCSOLVER_STATUS_NOT_INITIALIZED	库未初始化。
MCSOLVER_STATUS_INVALID_VALUE	传递了无效参数 (m,n<0 或错误的 lda)。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.3.3 mcsolverDn<t>sytrd()

这些辅助函数计算所需的工作缓冲区的大小。

```
mcsolverStatus_t
mcsolverDnSsytrd_bufferSize(
    mcsolverDnHandle_t handle,
    mcblasFillMode_t uplo,
    int n,
    const float *A,
    int lda,
    const float *d,
    const float *e,
    const float *tau,
    int *lwork);

mcsolverStatus_t
mcsolverDnDsytrd_bufferSize(
    mcsolverDnHandle_t handle,
    mcblasFillMode_t uplo,
    int n,
    const double *A,
    int lda,
    const double *d,
    const double *e,
    const double *tau,
    int *lwork);

mcsolverStatus_t
mcsolverDnChetrd_bufferSize(
    mcsolverDnHandle_t handle,
    mcblasFillMode_t uplo,
    int n,
    const mcComplex *A,
    int lda,
    const float *d,
    const float *e,
    const mcComplex *tau,
    int *lwork);

mcsolverStatus_t
mcsolverDnZhetrd_bufferSize(
    mcsolverDnHandle_t handle,
    mcblasFillMode_t uplo,
    int n,
    const mcDoubleComplex *A,
    int lda,
    const double *d,
    const double *e,
    const mcDoubleComplex *tau,
    int *lwork);
```

S 和 D 数据类型分别表示单精度实数和双精度实数。

```
mcsolverStatus_t
mcsolverDnSsytrd(
    mcsolverDnHandle_t handle,
    mcblasFillMode_t uplo,
```

(下页继续)

(续上页)

```
    int n,
    float *A,
    int lda,
    float *d,
    float *e,
    float *tau,
    float *work,
    int lwork,
    int *devInfo);

mcsolverStatus_t
mcsolverDnDsytrd(
    mcsolverDnHandle_t handle,
    mcblasFillMode_t uplo,
    int n,
    double *A,
    int lda,
    double *d,
    double *e,
    double *tau,
    double *work,
    int lwork,
    int *devInfo);
```

C 和 Z 数据类型分别表示单精度复数和双精度复数。

```
mcsolverStatus_t
mcsolverDnChetrd(
    mcsolverDnHandle_t handle,
    mcblasFillMode_t uplo,
    int n,
    mcComplex *A,
    int lda,
    float *d,
    float *e,
    mcComplex *tau,
    mcComplex *work,
    int lwork,
    int *devInfo);

mcsolverStatus_t mcsolverDnZhetrd(
    mcsolverDnHandle_t handle,
    mcblasFillMode_t uplo,
    int n,
    mcDoubleComplex *A,
    int lda,
    double *d,
    double *e,
    mcDoubleComplex *tau,
    mcDoubleComplex *work,
    int lwork,
    int *devInfo);
```

这个函数通过正交变换 $Q^H * A * Q = T$ 将一个一般的对称 (厄米特) n×n 矩阵 A 简化为一个实数对称三对角形式的矩阵 T。

作为输出，A 包含 T 和 Householder 反射向量。如果 uplo = MCBLAS_FILL_MODE_UPPER，则 A 的对角线和第一个超对角线会被三对角矩阵 T 对应的元素覆盖，而位于第一个超对角线以上的元素，通过数

组 tau 表示正交矩阵 Q 为一系列初等反射的乘积；如果 uplo = MCBLAS_FILL_MODE_LOWER，则 A 的对角线和第一个次对角线会被三对角矩阵 T 对应的元素覆盖，而位于第一个次对角线以下的元素，通过数组 tau 表示正交矩阵 Q 为一系列初等反射的乘积。

用户必须提供输入参数 work 所指向的工作空间。输入参数 lwork 表示工作空间的大小，可以通过 sytrd_bufferSize() 函数返回。如果输出参数 devInfo = -i （小于 0），则表示第 i 个参数是错误的（不包含句柄）。

2.4.3.4 mcsolverDn<t>ormtr()

这些辅助函数计算所需的工作缓冲区的大小。

```
mcsolverStatus_t
mcsolverDnSormtr_bufferSize(
    mcsolverDnHandle_t handle,
    mcblasSideMode_t side,
    mcblasFillMode_t uplo,
    mcblasOperation_t trans,
    int m,
    int n,
    const float *A,
    int lda,
    const float *tau,
    const float *C,
    int ldc,
    int *lwork);

mcsolverStatus_t
mcsolverDnDormtr_bufferSize(
    mcsolverDnHandle_t handle,
    mcblasSideMode_t side,
    mcblasFillMode_t uplo,
    mcblasOperation_t trans,
    int m,
    int n,
    const double *A,
    int lda,
    const double *tau,
    const double *C,
    int ldc,
    int *lwork);

mcsolverStatus_t
mcsolverDnCunmtr_bufferSize(
    mcsolverDnHandle_t handle,
    mcblasSideMode_t side,
    mcblasFillMode_t uplo,
    mcblasOperation_t trans,
    int m,
    int n,
    const mcComplex *A,
    int lda,
    const mcComplex *tau,
    const mcComplex *C,
    int ldc,
    int *lwork);
```

(下页继续)

(续上页)

```
mcsolverStatus_t
mcsolverDnZunmtr_bufferSize(
    mcsolverDnHandle_t handle,
    mcblasSideMode_t side,
    mcblasFillMode_t uplo,
    mcblasOperation_t trans,
    int m,
    int n,
    const mcDoubleComplex *A,
    int lda,
    const mcDoubleComplex *tau,
    const mcDoubleComplex *C,
    int ldc,
    int *lwork);
```

S 和 D 数据类型分别表示单精度实数和双精度实数。

```
mcsolverStatus_t
mcsolverDnSormtr(
    mcsolverDnHandle_t handle,
    mcblasSideMode_t side,
    mcblasFillMode_t uplo,
    mcblasOperation_t trans,
    int m,
    int n,
    float *A,
    int lda,
    float *tau,
    float *C,
    int ldc,
    float *work,
    int lwork,
    int *devInfo);

mcsolverStatus_t
mcsolverDnDormtr(
    mcsolverDnHandle_t handle,
    mcblasSideMode_t side,
    mcblasFillMode_t uplo,
    mcblasOperation_t trans,
    int m,
    int n,
    double *A,
    int lda,
    double *tau,
    double *C,
    int ldc,
    double *work,
    int lwork,
    int *devInfo);
```

C 和 Z 数据类型分别表示单精度复数和双精度复数。

```
mcsolverStatus_t
mcsolverDnCunmtr(
    mcsolverDnHandle_t handle,
    mcblasSideMode_t side,
```

(下页继续)

(续上页)

```
    mcblasFillMode_t uplo,
    mcblasOperation_t trans,
    int m,
    int n,
    mcComplex *A,
    int lda,
    mcComplex *tau,
    mcComplex *C,
    int ldc,
    mcComplex *work,
    int lwork,
    int *devInfo);

mcsolverStatus_t
mcsolverDnZunmtr(
    mcsolverDnHandle_t handle,
    mcblasSideMode_t side,
    mcblasFillMode_t uplo,
    mcblasOperation_t trans,
    int m,
    int n,
    mcDoubleComplex *A,
    int lda,
    mcDoubleComplex *tau,
    mcDoubleComplex *C,
    int ldc,
    mcDoubleComplex *work,
    int lwork,
    int *devInfo);
```

Q 是由一系列初等反射向量形成的一个酉矩阵，初等反射向量来自 sytrd。

用户必须提供输入参数 work 所指向的工作空间。输入参数 lwork 表示工作空间的大小，可以通过 ormtr_bufferSize() 函数返回。如果输出参数 devInfo = -i (小于 0)，则表示第 i 个参数是错误的 (不包含句柄)。

API of ormtr

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库上下文的句柄。
side	host	input	side = MCBLAS_SIDE_LEFT：从左侧应用 Q 或 Q**T；side = MCBLAS_SIDE_RIGHT：从右侧应用 Q 或 Q**T。
uplo	host	input	uplo = MCBLAS_FILL_MODE_LOWER：矩阵 A 的下三角包含从 sytrd 返回的基本反射器。uplo = MCBLAS_FILL_MODE_UPPER：矩阵 A 的上三角包含从 sytrd 返回的基本反射器。
trans	host	input	非转置或（共轭）转置的操作 op(Q)。
m	host	input	矩阵 C 的行数。
n	host	input	矩阵 C 的列数。
A	device	in/out	<type> 尺寸为 lda * m（如果 side = MCBLAS_SIDE_LEFT）；尺寸为 lda * n（如果 side = MCBLAS_SIDE_RIGHT）的数组。矩阵 A 来自 sytrd，并包含基本反射器。
lda	host	input	用于存储矩阵 A 的二维数组前导维度。如果 side 为 MCBLAS_SIDE_LEFT，则 lda >= max(1,m)；如果 side 为 MCBLAS_SIDE_RIGHT，则 lda >= max(1,n)。

下页继续

表 2.17 – 续上页

参数	内存	In/out	含义
tau	device	output	<type> 尺寸为 (m-1) （如果 side = MCBLAS_SIDE_LEFT）；尺寸为 (n-1) （如果 side = MCBLAS_SIDE_RIGHT）的数组。向量 tau 来自 sytrd，因此 tau(i) 是第 i 个基本反射器向量的标量。
C	device	in/out	<type> 尺寸为 ldc * n 的数组。退出时，C 被 op(Q)*C 或 C*op(Q) 覆盖。
ldc	host	input	用于存储矩阵 C 的二维数组前导维度。ldc >= max(1,m)。
work	device	in/out	工作空间，<type> 大小为 lwork 的数组。
lwork	host	input	工作数组 work 的大小。
devInfo	device	output	如果 devInfo = 0，则 ormqr 成功。如果 devInfo = -i，第 i 个参数不正确（不计入句柄）。

返回状态

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。
MCSOLVER_STATUS_NOT_INITIALIZED	库未进行初始化
MCSOLVER_STATUS_INVALID_VALUE	传递的参数无效 (n<0 或 lda<max(1,n))。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.3.5 mcsolverDn<t>orgtr()

这些辅助函数计算所需的工作缓冲区的大小。

```
mcsolverStatus_t
mcsolverDnSorgtr_bufferSize(
    mcsolverDnHandle_t handle,
    mcblasFillMode_t uplo,
    int n,
    const float *A,
    int lda,
    const float *tau,
    int *lwork);

mcsolverStatus_t
mcsolverDnDorgtr_bufferSize(
    mcsolverDnHandle_t handle,
    mcblasFillMode_t uplo,
    int n,
    const double *A,
    int lda,
    const double *tau,
    int *lwork);

mcsolverStatus_t
mcsolverDnCungtr_bufferSize(
    mcsolverDnHandle_t handle,
    mcblasFillMode_t uplo,
    int n,
    const mcComplex *A,
    int lda,
    const mcComplex *tau,
    int *lwork);
```

(下页继续)

(续上页)

```
mcsolverStatus_t
mcsolverDnZungtr_bufferSize(
    mcsolverDnHandle_t handle,
    mcblasFillMode_t uplo,
    int n,
    const mcDoubleComplex *A,
    int lda,
    const mcDoubleComplex *tau,
    int *lwork);
```

S 和 D 数据类型分别表示单精度实数和双精度实数。

```
mcsolverStatus_t
mcsolverDnSorgtr(
    mcsolverDnHandle_t handle,
    mcblasFillMode_t uplo,
    int n,
    float *A,
    int lda,
    const float *tau,
    float *work,
    int lwork,
    int *devInfo);

mcsolverStatus_t
mcsolverDnDorgtr(
    mcsolverDnHandle_t handle,
    mcblasFillMode_t uplo,
    int n,
    double *A,
    int lda,
    const double *tau,
    double *work,
    int lwork,
    int *devInfo);
```

C 和 Z 数据类型分别表示单精度复数和双精度复数。

```
mcsolverStatus_t
mcsolverDnCungtr(
    mcsolverDnHandle_t handle,
    mcblasFillMode_t uplo,
    int n,
    mcComplex *A,
    int lda,
    const mcComplex *tau,
    mcComplex *work,
    int lwork,
    int *devInfo);

mcsolverStatus_t
mcsolverDnZungtr(
    mcsolverDnHandle_t handle,
    mcblasFillMode_t uplo,
    int n,
    mcDoubleComplex *A,
```

(下页继续)

(续上页)

```
    int lda,
    const mcDoubleComplex *tau,
    mcDoubleComplex *work,
    int lwork,
    int *devInfo);
```

这个函数生成一个酉矩阵 Q，它被定义为 n-1 个阶为 n 的初等反射向量的乘积，由 sytrd 返回。

用户必须提供输入参数 work 所指向的工作空间。输入参数 lwork 表示工作空间的大小，由 orgtr_bufferSize() 函数返回。

如果输出参数 devInfo = -i (小于 0)，则表示第 i 个参数是错误的 (不包含句柄)。

返回状态

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。
MCSOLVER_STATUS_NOT_INITIALIZED	库未进行初始化。
MCSOLVER_STATUS_INVALID_VALUE	传递的参数无效 (n<0 或 lda<max(1,n))。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.3.6 mcsolverDn<t>gesvd()

这些辅助函数计算所需的工作缓冲区的大小。

```
mcsolverStatus_t
mcsolverDnSgesvd_bufferSize(
    mcsolverDnHandle_t handle,
    int m,
    int n,
    int *lwork );

mcsolverStatus_t
mcsolverDnDgesvd_bufferSize(
    mcsolverDnHandle_t handle,
    int m,
    int n,
    int *lwork );

mcsolverStatus_t
mcsolverDnCgesvd_bufferSize(
    mcsolverDnHandle_t handle,
    int m,
    int n,
    int *lwork );

mcsolverStatus_t
mcsolverDnZgesvd_bufferSize(
    mcsolverDnHandle_t handle,
    int m,
    int n,
    int *lwork );
```

S 和 D 数据类型分别表示单精度实数和双精度实数。

```
mcsolverStatus_t
mcsolverDnSgesvd (
    mcsolverDnHandle_t handle,
    signed char jobu,
    signed char jobvt,
    int m,
    int n,
    float *A,
    int lda,
    float *S,
    float *U,
    int ldu,
    float *VT,
    int ldvt,
    float *work,
    int lwork,
    float *rwork,
    int *devInfo);

mcsolverStatus_t
mcsolverDnDgesvd (
    mcsolverDnHandle_t handle,
    signed char jobu,
    signed char jobvt,
    int m,
    int n,
    double *A,
    int lda,
    double *S,
    double *U,
    int ldu,
    double *VT,
    int ldvt,
    double *work,
    int lwork,
    double *rwork,
    int *devInfo);
```

C 和 Z 数据类型分别表示单精度复数和双精度复数。

```
mcsolverStatus_t
mcsolverDnCgesvd (
    mcsolverDnHandle_t handle,
    signed char jobu,
    signed char jobvt,
    int m,
    int n,
    mcComplex *A,
    int lda,
    float *S,
    mcComplex *U,
    int ldu,
    mcComplex *VT,
    int ldvt,
    mcComplex *work,
    int lwork,
    float *rwork,
```

(下页继续)

(续上页)

```
    int *devInfo);

mcsolverStatus_t
mcsolverDnZgesvd (
    mcsolverDnHandle_t handle,
    signed char jobu,
    signed char jobvt,
    int m,
    int n,
    mcDoubleComplex *A,
    int lda,
    double *S,
    mcDoubleComplex *U,
    int ldu,
    mcDoubleComplex *VT,
    int ldvt,
    mcDoubleComplex *work,
    int lwork,
    double *rwork,
    int *devInfo);
```

此函数计算 m×n 矩阵 A 的奇异值分解（SVD）以及对应的左奇异和/或右奇异向量。SVD 的写法如下

$$A = U * \Sigma * V^H$$

在这里，Σ 是一个 m×n 矩阵，除了其 min(m,n) 个对角线元素外，其它元素都为零。U 是一个 m×m 的酉矩阵，V 是一个 n×n 的酉矩阵。Σ 的对角线元素是矩阵 A 的奇异值；它们是实数且非负，并按降序返回。U 和 V 的前 min(m,n) 列是矩阵 A 的左奇异向量和右奇异向量。

用户必须提供输入参数 work 所指向的工作空间。输入参数 lwork 表示工作空间的大小，可以通过 gesvd_bufferSize() 函数返回。

如果输出参数 devInfo = -i (小于 0)，则表示第 i 个参数是错误的 (不包含句柄)。如果 bdsqr 没有收敛，devInfo 会指定有多少个中间双对角形式的超对角线没有收敛为零。

如果 devInfo > 0 并且 rwork 不为空，则 rwork 是一个维度为 (min(m,n)-1) 的实数数组，其中包含未收敛的上三角双对角矩阵的超对角线元素。这与 LAPACK 稍有不同，如果类型是 real ，则未收敛的超对角线元素会放在 work 中；如果类型是 complex ，则会放在 rwork 中。如果用户不需要超对角线的信息，rwork 可以是一个空指针（即为 NULL）。

备注 1：gesvd 函数仅支持 m>=n 。

备注 2：该例程返回的是 V^H，而不是 V 。

API of gesvd

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库的句柄。
jobu	host	input	对于计算矩阵 U 全部或部分选项的说明：值为 A 时，将 U 的所有 m 列返回到数组 U 中。值为 S 时，将 U 的前 min(m,n) 列（左奇异向量）返回到数组 U 中。值为 O 时，将 U 的前 min(m,n) 列（左奇异向量）覆写在数组 A 上。值为 N 时，不计算 U 的任何列（没有左奇异向量）。
jobvt	host	input	对于计算矩阵 V**T 的全部或部分选项进行说明：值为 A 时，将 V**T 的所有 N 行返回到数组 VT 中。值为 S 时，将 V**T 的前 min(m,n) 行（右奇异向量）返回到数组 VT 中。值为 O 时，将 V**T 的前 min(m,n) 行（右奇异向量）覆写在数组 A 上。值为 N 时，不计算 V**T 的任何行（没有右奇异向量）。
m	host	input	矩阵 A 的行数。

下页继续

表 2.18 – 续上页

参数	内存	In/out	含义
n	host	input	矩阵 A 的列数。
A	device	in/out	<type> 数组的维度为 lda*n，其中 lda 不小于 max(1,m)。退出时，A 中的内容将被清空。
lda	host	input	用于存储矩阵 A 的二维数组的前导维度。
S	device	output	维度为 min(m,n) 的实数数组。矩阵 A 的奇异值按照 S(i) >= S(i+1) 的顺序进行排序。
U	device	output	<type> 数组的维度为 ldu*m，其中 ldu 不小于 max(1,m)。U 包含了 m×m 的酉矩阵 U。
ldu	host	input	用于存储矩阵 U 的二维数组的前导维度。
VT	device	output	维度为 ldvt*n 的 <type> 数组，其中 ldvt 不小于 max(1,n)。数组 VT 包含了一个 n×n 的酉矩阵 V**T。
ldvt	host	input	用于存储矩阵 Vt 的二维数组的前导维度。
work	device	in/out	存储于大小为 lwork 的 <type> 数组的工作空间。
lwork	host	input	gesvd_bufferSize 函数返回的 work 大小。
rwork	device	input	维度为 min(m,n)-1 的实数数组。如果 devInfo > 0，则说明其包含上半双对角矩阵的未收敛超对角元素。
devInfo	device	output	如果 devInfo=0，则操作成功。如果 devInfo=-i，则第 i 个参数是错误的 (不包括计数句柄)。如果 devInfo > 0，则 devInfo 说明有多少个超对角线的中间双对角形式未收敛为零。

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。
MCSOLVER_STATUS_NOT_INITIALIZED	库未进行初始化。
MCSOLVER_STATUS_INVALID_VALUE	传递的参数无效 (n<0 或 lda<max(1,n))。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.3.7 mcsolverDnGesvd()

下面的辅助函数可以计算所需的预分配缓冲区大小。

```
mcsolverStatus_t mcsolverDnGesvd_bufferSize(
    mcsolverDnHandle_t handle,
    mcsolverDnParams_t params,
    signed char jobu,
    signed char jobvt,
    int64_t m,
    int64_t n,
    macaDataType dataTypeA,
    const void *A,
    int64_t lda,
    macaDataType dataTypeS,
    const void *S,
    macaDataType dataTypeU,
    const void *U,
    int64_t ldu,
    macaDataType dataTypeVT,
    const void *VT,
    int64_t ldvt,
    macaDataType computeType,
    size_t *workspaceInBytes);
```

下面的例程:

```
mcsolverStatus_t mcsolverAPI mcsolverDnGesvd(
    mcsolverDnHandle_t handle,
    mcsolverDnParams_t params,
    signed char jobu,
    signed char jobvt,
    int64_t m,
    int64_t n,
    macaDataType dataTypeA,
    void *A,
    int64_t lda,
    macaDataType dataTypeS,
    void *S,
    macaDataType dataTypeU,
    void *U,
    int64_t ldu,
    macaDataType dataTypeVT,
    void *VT,
    int64_t ldvt,
    macaDataType computeType,
    void *pBuffer,
    size_t workspaceInBytes,
    int *info);
```

此函数计 m×n 矩阵 A 的奇异值分解（SVD）以及对应的左奇异和/或右奇异向量。SVD 的写法如下

$$A = U * \Sigma * V^H$$

其中 Σ 是一个 m×n 矩阵，除了其 min(m,n) 个对角元素外，其余元素均为零。U 是一个 m×m 酉矩阵。V 是一个 n×n 酉矩阵。Σ 的对角元素是 A 的奇异值；它们是实数非负数，并按降序返回。U 和 V 的前 min(m,n) 列分别是 A 的左右奇异向量。

用户需要提供由输入参数 pBuffer 指向的工作空间。输入参数 workspaceInBytes 表示工作空间的大小（以字节为单位），它由 mcsolverDnGesvd_bufferSize() 返回。

如果输出参数 info=-i (小于 0)，则第 i 个参数是错误的 (不包含句柄)。如果 bdsqr 没有收敛，info 指定有多少中间双对角线形式的超对角线不收敛于零。目前，mcsolverDnGesvd 仅支持默认的算法。

mcsolverDnGesvd 支持的算法表

mcsolver_ALG_0 or NULL	默认算法

备注 1: gesvd 仅支持 m>=n 。

备注 2: 例程返回 V^H，而不是 V。

API of mcsolverDnGesvd

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库的句柄。
params	host	input	由 mcsolverDnSetAdvOptions 函数收集的信息结构。
jobu	host	input	对于计算矩阵 U 全部或部分选项的说明：值为 A 时，将 U 的所有 m 列返回到数组 U 中。值为 S 时，将 U 的前 min(m,n) 列（左奇异向量）返回到数组 U 中。值为 O 时，将 U 的前 min(m,n) 列（左奇异向量）覆写在数组 A 上。值为 N 时，不计算 U 的任何列（没有左奇异向量）。

下页继续

表 2.19 – 续上页

参数	内存	In/out	含义
jobvt	host	input	对于计算矩阵 V**T 的全部或部分选项进行说明：值为 A 时，将 V**T 的所有 N 行返回到数组 VT 中。值为 S 时，将 V**T 的前 min(m,n) 行（右奇异向量）返回到数组 VT 中。值为 O 时，将 V**T 的前 min(m,n) 行（右奇异向量）覆写在数组 A 上。值为 N 时，不计算 V**T 的任何行（没有右奇异向量）。
m	host	input	矩阵 A 的行数。
n	host	input	矩阵 A 的列数。
dataTypeA	host	input	数组 A 的数据类型。
A	device	in/out	数组的维度为 lda * n，其中 lda 不小于 max(1,m)。退出时，A 中的内容将被清空。
lda	host	input	用于存储矩阵 A 的二维数组的前导维度。
dataTypeS	host	input	数组 S 的数据类型。
S	device	output	维度为 min(m,n) 的实数数组。矩阵 A 的奇异值按照 S(i)` >= S(i+1) 的顺序进行排序。
dataTypeU	host	input	数组 U 的数据类型。
U	device	output	数组的维度为 ldu * m，其中 ldu 不小于 max(1,m)。U 包含了 m×m 的酉矩阵 U。
ldu	host	input	用于存储矩阵 U 的二维数组的前导维度。
dataTypeVT	host	input	数组 VT 的数据类型。
VT	device	output	维度为 ldvt * n 的数组，其中 ldvt 不小于 max(1, n)。数组 VT 包含了一个 n×n 的酉矩阵 V**T。
ldvt	host	input	用于存储矩阵 Vt 的二维数组的前导维度。
computeType	host	input	计算数据类型。
pBuffer	device	in/out	存储于大小为 workspaceInBytes 字节的 void 类型数组的工作空间。
workspaceInBytes	host	input	由 mcsolverDnGesvd_bufferSize 返回的以字节为单位的 pBuffer 大小。
info	device	output	如果 info=0，则操作成功。如果 info = -i，则第 i 个参数是错误的 (不包括计数句柄)。如果 info > 0，则 info 说明有多少个超对角线的中间双对角形式未收敛为零。

通用 API 有三种不同类型：

- dataTypeA 为矩阵 A 的数据类型
- dataTypeS 是向量 S 的数据类型且 dataTypeU 是矩阵 U 的数据类型
- dataTypeVT 是矩阵 VT 的数据类型，computeType 是运算的计算类型

mcsolverDnGesvd 仅支持以下四种组合。

数据类型和计算类型的有效组合

DataTypeA	DataTypeS	DataTypeU	DataTypeVT	ComputeType	Meaning
MACA_R_32F	MACA_R_32F	MACA_R_32F	MACA_R_32F	MACA_R_32F	SGESVD
MACA_R_64F	MACA_R_64F	MACA_R_64F	MACA_R_64F	MACA_R_64F	DGESVD
MACA_C_32F	MACA_R_32F	MACA_C_32F	MACA_C_32F	MACA_C_32F	CGESVD
MACA_C_64F	MACA_R_64F	MACA_C_64F	MACA_C_64F	MACA_C_64F	ZGESVD

返回状态

MCSOLVER_STATUS_SUCCESS	操作成功完成。
MCSOLVER_STATUS_NOT_INITIALIZED	库未初始化。
MCSOLVER_STATUS_INVALID_VALUE	传递了无效参数：m,n<0 ，lda<max(1,m) ，ldu<max(1,m) 或 ldvt<max(1,n)。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.3.8 mcsolverDn<t>gesvdj()

下面的辅助函数可以计算所需的预分配缓冲区大小。

```
mcsolverStatus_t
mcsolverDnSgesvdj_bufferSize(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    int econ,
    int m,
    int n,
    const float *A,
    int lda,
    const float *S,
    const float *U,
    int ldu,
    const float *V,
    int ldv,
    int *lwork,
    gesvdjInfo_t params);

mcsolverStatus_t
mcsolverDnDgesvdj_bufferSize(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    int econ,
    int m,
    int n,
    const double *A,
    int lda,
    const double *S,
    const double *U,
    int ldu,
    const double *V,
    int ldv,
    int *lwork,
    gesvdjInfo_t params);

mcsolverStatus_t
mcsolverDnCgesvdj_bufferSize(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    int econ,
    int m,
    int n,
    const mcComplex *A,
    int lda,
    const float *S,
    const mcComplex *U,
```

(下页继续)

(续上页)

```
    int ldu,
    const mcComplex *V,
    int ldv,
    int *lwork,
    gesvdjInfo_t params);

mcsolverStatus_t
mcsolverDnZgesvdj_bufferSize(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    int econ,
    int m,
    int n,
    const mcDoubleComplex *A,
    int lda,
    const double *S,
    const mcDoubleComplex *U,
    int ldu,
    const mcDoubleComplex *V,
    int ldv,
    int *lwork,
    gesvdjInfo_t params);
```

S 和 D 数据类型分别表示单精度实数和双精度实数。

```
mcsolverStatus_t
mcsolverDnSgesvdj(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    int econ,
    int m,
    int n,
    float *A,
    int lda,
    float *S,
    float *U,
    int ldu,
    float *V,
    int ldv,
    float *work,
    int lwork,
    int *info,
    gesvdjInfo_t params);

mcsolverStatus_t
mcsolverDnDgesvdj(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    int econ,
    int m,
    int n,
    double *A,
    int lda,
    double *S,
    double *U,
    int ldu,
```

(下页继续)

(续上页)

```
    double *V,
    int ldv,
    double *work,
    int lwork,
    int *info,
    gesvdjInfo_t params);
```

C 和 Z 数据类型分别表示单精度复数和双精度复数。

```
mcsolverStatus_t
mcsolverDnCgesvdj(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    int econ,
    int m,
    int n,
    mcComplex *A,
    int lda,
    float *S,
    mcComplex *U,
    int ldu,
    mcComplex *V,
    int ldv,
    mcComplex *work,
    int lwork,
    int *info,
    gesvdjInfo_t params);

mcsolverStatus_t
mcsolverDnZgesvdj(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    int econ,
    int m,
    int n,
    mcDoubleComplex *A,
    int lda,
    double *S,
    mcDoubleComplex *U,
    int ldu,
    mcDoubleComplex *V,
    int ldv,
    mcDoubleComplex *work,
    int lwork,
    int *info,
    gesvdjInfo_t params);
```

此函数计算 `m×n` 矩阵 `A` 的奇异值分解（SVD）以及对应的左奇异和/或右奇异向量。SVD 的写法如下：

$$A = U * \Sigma * V^H$$

其中 Σ 是一个 `m×n` 矩阵，除了其 `min(m,n)` 个对角元素外，其余元素均为零。`U` 是一个 `m×m` 酉矩阵。`V` 是一个 `n×n` 酉矩阵。Σ 的对角元素是 `A` 的奇异值；它们是实数非负数，并按降序返回。`U` 和 `V` 的前 `min(m,n)` 列分别是 `A` 的左右奇异向量。

`gesvdj` 与 `gesvd` 具有相同的功能。区别在于 `gesvd` 使用 QR 算法，而 `gesvdj` 使用 Jacobi 方法。Jacobi 方法的并行性使 GPU 在处理中小型矩阵时具有更好的性能。此外，用户可以通过配置 `gesvdj` 来实现达到一定精度的近似值。

gesvdj 迭代生成一个酉矩阵序列，将矩阵 A 转换为以下形式

$$U^H * A * V = S + E$$

其中 S 为对角线且 E 的对角线为零。

在迭代过程中，E 的 Frobenius 范数单调递减。当 E 趋近于 0 时，S 是奇异值的集合。在实践中，Jacobi 方法停止迭代。

gesvdj 有两个参数来控制精度。第一个参数是公差 (eps)。默认值为机器精度，但用户可以使用 mcsolverDnXgesvdjSetTolerance 函数设置先验公差。第二个参数是最大扫描次数，控制 Jacobi 方法的迭代次数。默认值为 100 但用户可以使用 mcsolverDnXgesvdjSetMaxSweeps 函数设置一个适当的边界。实验表明，15 次扫描足以收敛到机器精度。gesvdj 在满足公差或满足最大扫描次数时停止。

Jacobi 方法具有二次收敛性，因此精度与扫描次数不成正比。为保证一定的精度，用户应只配置公差。

用户必须提供输入参数 work 所指向的工作空间。输入参数 lwork 表示工作空间的大小，并且由 gesvdj_bufferSize() 返回。

如果输出参数 info=-i (小于 0)，则第 i 个参数是错误的 (不包含句柄)。

如果 info=min(m,n)+1，gesvdj 在给定容差和最大扫描下不收敛。

如果用户设置的公差不合适，gesvdj 可能不会收敛。例如，公差不应小于机器精度。

备注 1: gesvdj 支持 m 和 n 的任意组合。

备注 2: 例程返回 V，而不是 V^H。这与 gesvd 不同。

API of gesvdj

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库的句柄。
jobz	host	input	该参数用于指定计算奇异值还是同时计算奇异值和奇异向量: jobz = MCSOLVER_EIG_MODE_NOVECTOR：表示仅计算奇异值; jobz = MCSOLVER_EIG_MODE_VECTOR：表示同时计算奇异值和奇异向量。
econ	host	input	参数 econ = 1 表示在计算矩阵 U 和 V 时使用经济尺寸。
m	host	input	矩阵 A 的行数。
n	host	input	矩阵 A 的列数。
A	device	in/out	<type> 数组的维度为 lda*n，其中 lda 不小于 max(1,m)。退出时，A 中的内容将被清空。
lda	host	input	用于存储矩阵 A 的二维数组的前导维度。
S	device	output	维度为 min(m,n) 的实数数组。矩阵 A 的奇异值按照 S(i) >= S(i+1) 的顺序进行排序。
U	device	output	如果 econ 为零，则 <type> 数组的维度为 ldu * m。如果 econ 参数非零，则维度为 ldu * min(m,n)。U 包含左奇异向量。
ldu	host	input	用于存储矩阵 U 的二维数组的前导维度。ldu 不小于 max(1,m)。
V	device	output	如果 econ 为零，则 <type> 数组的维度为 ldv * n。如果 econ 参数非零，则维度为 ldv * min(m,n)。V 包含左奇异向量。
ldv	host	input	用于存储矩阵 V 的二维数组的前导维度。ldv 不小于 max(1,n)。
work	device	in/out	用作工作空间大小为 lwork 的 <type> 数组。
lwork	host	input	gesvd_bufferSize 函数返回的 work 大小。
info	device	output	如果 info=0，则操作成功。如果 info = -i，则第 i 个参数是错误的 (不包括计数句柄)。如果 info = min(m,n)+1，则表示在给定的容差和最大迭代次数下 gesvdj 未收敛。
params	host	in/out	该结构包含了 Jacobi 算法的参数以及 gesvdj 的结果。

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。
MCSOLVER_STATUS_NOT_INITIALIZED	库未进行初始化
MCSOLVER_STATUS_INVALID_VALUE	传递的参数无效 (n<0 或 lda<max(1,n))。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.3.9 mcsolverDn<t>gesvdjBatched()

下面的辅助函数可以计算预分配缓冲区所需的大小。

```
mcsolverStatus_t
mcsolverDnSgesvdjBatched_bufferSize(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    int m,
    int n,
    const float *A,
    int lda,
    const float *S,
    const float *U,
    int ldu,
    const float *V,
    int ldv,
    int *lwork,
    gesvdjInfo_t params,
    int batchSize);

mcsolverStatus_t
mcsolverDnDgesvdjBatched_bufferSize(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    int m,
    int n,
    const double *A,
    int lda,
    const double *S,
    const double *U,
    int ldu,
    const double *V,
    int ldv,
    int *lwork,
    gesvdjInfo_t params,
    int batchSize);

mcsolverStatus_t
mcsolverDnCgesvdjBatched_bufferSize(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    int m,
    int n,
    const mcComplex *A,
    int lda,
    const float *S,
    const mcComplex *U,
    int ldu,
```

(下页继续)

(续上页)

```
    const mcComplex *V,
    int ldv,
    int *lwork,
    gesvdjInfo_t params,
    int batchSize);

mcsolverStatus_t
mcsolverDnZgesvdjBatched_bufferSize(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    int m,
    int n,
    const mcDoubleComplex *A,
    int lda,
    const double *S,
    const mcDoubleComplex *U,
    int ldu,
    const mcDoubleComplex *V,
    int ldv,
    int *lwork,
    gesvdjInfo_t params,
    int batchSize);
```

S 和 D 数据类型分别表示单精度和双精度实数。

```
mcsolverStatus_t
mcsolverDnSgesvdjBatched(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    int m,
    int n,
    float *A,
    int lda,
    float *S,
    float *U,
    int ldu,
    float *V,
    int ldv,
    float *work,
    int lwork,
    int *info,
    gesvdjInfo_t params,
    int batchSize);

mcsolverStatus_t
mcsolverDnDgesvdjBatched(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    int m,
    int n,
    double *A,
    int lda,
    double *S,
    double *U,
    int ldu,
    double *V,
```

(下页继续)

(续上页)

```
    int ldv,
    double *work,
    int lwork,
    int *info,
    gesvdjInfo_t params,
    int batchSize);
```

C 和 Z 数据类型分别表示单精度和双精度复数。

```
mcsolverStatus_t
mcsolverDnCgesvdjBatched(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    int m,
    int n,
    mcComplex *A,
    int lda,
    float *S,
    mcComplex *U,
    int ldu,
    mcComplex *V,
    int ldv,
    mcComplex *work,
    int lwork,
    int *info,
    gesvdjInfo_t params,
    int batchSize);

mcsolverStatus_t
mcsolverDnZgesvdjBatched(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    int m,
    int n,
    mcDoubleComplex *A,
    int lda,
    double *S,
    mcDoubleComplex *U,
    int ldu,
    mcDoubleComplex *V,
    int ldv,
    mcDoubleComplex *work,
    int lwork,
    int *info,
    gesvdjInfo_t params,
    int batchSize);
```

该函数计算一系列一般 m×n 矩阵的奇异值和奇异向量。

该矩阵是一个实数 m×n 对角矩阵，除了其 min(m,n) 个对角元素外，其余元素均为零。(左奇异向量) 是一个 m×m 的酉矩阵，(右奇异向量) 是一个 n×n 的酉矩阵。Σ_j 中奇异值，可以按降序或非排序顺序排列。

gesvdjBatched 对每个矩阵执行 gesvdj 操作。它要求所有矩阵的大小都同为 m,n 且不超过 32 并以连续方式打包。

每个矩阵都是以列为主的，且前导维度为 lda ，因此随机访问的公式为。

参数 S 也以连续方式包含了每个矩阵的奇异值。

$$S = \begin{pmatrix} S0 & S1 & \cdots \end{pmatrix}$$

S 的随机访问公式是。

除了容差和最大迭代次数之外，函数 gesvdjBatched 可以通过函数 mcsolverDnXgesvdjSetSortEig 以降序（默认）或原样（不排序）的方式对奇异值进行排序。如果用户将多个小矩阵打包成一个矩阵的对角块，那么非排序选项可以将这些小矩阵的奇异值分开。

gesvdjBatched 无法通过函数 mcsolverDnXgesvdjGetResidual 和 mcsolverDnXgesvdjGetSweeps 报告残差和执行的迭代次数。对这两个函数的任何调用都将返回 MCSOLVER_STATUS_NOT_SUPPORTED。因此用户需要显式计算残差。

用户必须使用输入参数 work 指定的工作空间。输入参数 lwork 表示工作空间的大小，并且可以通过 gesvdjBatched_bufferSize() 函数返回。

输出参数 info 是一个大小为 batchSize 的整数数组。如果函数返回 MCSOLVER_STATUS_INVALID_VALUE，则第一个元素 info[0] = -i （小于零）表示第 i 个参数错误（不包括句柄）。否则，如果 info[i] = min(m,n)+1，则 gesvdjBatched 在给定的容差和最大迭代次数下无法收敛于第 i 个矩阵。

API of syevjBatched

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库的句柄。
jobz	host	input	该参数用于指定计算奇异值还是同时计算奇异值和奇异向量: jobz = MCSOLVER_EIG_MODE_NOVECTOR：表示仅计算奇异值; jobz = MCSOLVER_EIG_MODE_VECTOR：表示同时计算奇异值和奇异向量。
m	host	input	矩阵 Aj 的行数，其中 m 不大于 32。
n	host	input	矩阵 Aj 的列数，其中 n 不大于 32。
A	device	in/out	<type> 数组的维度为 lda * n * batchSize，其中 lda 不小于 max(1,n)。退出时，Aj 中的内容将被清空。
lda	host	input	用于存储矩阵 Aj 的二维数组的前导维度。
S	device	output	一个维度为 min(m,n) * batchSize 的实数数组。它按降序或非排序顺序存储了矩阵 Aj 的奇异值。
U	device	output	维度为 ldu * m * batchSize 的 <type> 数组。Uj 包含了矩阵 Aj 的左奇异向量。
ldu	host	input	用于存储矩阵 Uj 的二维数组的前导维度。ldu 不小于 max(1,m)。
V	device	output	维度为 ldv * n * batchSize 的 <type> 数组。Vj 包含了矩阵 Aj 的左奇异向量。
ldv	host	input	用于存储矩阵 Vj 的二维数组的前导维度。ldv 不小于 max(1,n)。
work	device	in/out	用作工作空间大小为 lwork 的 <type> 数组。
lwork	host	input	gesvd_bufferSize 函数返回的 work 大小。
info	device	output	一个维度为 batchSize 的整数数组。如果返回 MCSOLVER_STATUS_INVALID_VALUE，则 info[0] = -i （小于零）表示第 i 个参数错误（不包括句柄）。否则，如果 info[i] = 0，则表示操作成功。如果 info[i] = min(m,n)+1，则在给定的容差和最大迭代次数下，gesvdjBatched 无法收敛于第 i 个矩阵。
params	host	in/out	填充有 Jacobi 算法参数的结构体。
batchSize	host	input	矩阵的数量。

返回状态

MCSOLVER_STATUS_SUCCESS	操作已成功完成。
MCSOLVER_STATUS_NOT_INITIALIZED	库未初始化。
MCSOLVER_STATUS_INVALID_VALUE	传递了无效的参数（m,n<0 或 lda<max(1,m) 或 ldu<max(1,m) 或 ldv<max(1,n) 或者 jobz 不是 MCSOLVER_EIG_MODE_NOVECTOR 或 MCSOLVER_EIG_MODE_VECTOR，或 batchSize<0）。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.3.10 mcsolverDn<t>gesvdaStridedBatched()

以下辅助函数可以计算预分配缓冲区所需的大小。

```
mcsolverStatus_t
mcsolverDnSgesvdaStridedBatched_bufferSize(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    int rank,
    int m,
    int n,
    const float *A,
    int lda,
    long long int strideA,
    const float *S,
    long long int strideS,
    const float *U,
    int ldu,
    long long int strideU,
    const float *V,
    int ldv,
    long long int strideV,
    int *lwork,
    int batchSize);

mcsolverStatus_t
mcsolverDnDgesvdaStridedBatched_bufferSize(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    int rank,
    int m,
    int n,
    const double *A,
    int lda,
    long long int strideA,
    const double *S,
    long long int strideS,
    const double *U,
    int ldu,
    long long int strideU,
    const double *V,
    int ldv,
    long long int strideV,
    int *lwork,
    int batchSize);
```

(下页继续)

(续上页)

```
mcsolverStatus_t
mcsolverDnCgesvdaStridedBatched_bufferSize(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    int rank,
    int m,
    int n,
    const mcComplex *A,
    int lda,
    long long int strideA,
    const float *S,
    long long int strideS,
    const mcComplex *U,
    int ldu,
    long long int strideU,
    const mcComplex *V,
    int ldv,
    long long int strideV,
    int *lwork,
    int batchSize);

mcsolverStatus_t
mcsolverDnZgesvdaStridedBatched_bufferSize(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    int rank,
    int m,
    int n,
    const mcDoubleComplex *A,
    int lda,
    long long int strideA,
    const double *S,
    long long int strideS,
    const mcDoubleComplex *U,
    int ldu,
    long long int strideU,
    const mcDoubleComplex *V,
    int ldv,
    long long int strideV,
    int *lwork,
    int batchSize);
```

S 和 D 数据类型分别表示单精度和双精度实数。

```
mcsolverStatus_t
mcsolverDnSgesvdaStridedBatched(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    int rank,
    int m,
    int n,
    const float *A,
    int lda,
    long long int strideA,
    float *S,
    long long int strideS,
```

(下页继续)

(续上页)

```
    float *U,
    int ldu,
    long long int strideU,
    float *V,
    int ldv,
    long long int strideV,
    float *work,
    int lwork,
    int *info,
    double *h_R_nrmF,
    int batchSize);

mcsolverStatus_t
mcsolverDnDgesvdaStridedBatched(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    int rank,
    int m,
    int n,
    const double *A,
    int lda,
    long long int strideA,
    double *S,
    long long int strideS,
    double *U,
    int ldu,
    long long int strideU,
    double *V,
    int ldv,
    long long int strideV,
    double *work,
    int lwork,
    int *info,
    double *h_R_nrmF,
    int batchSize);
```

C 和 Z 数据类型分别表示单精度和双精度复数。

```
mcsolverStatus_t
mcsolverDnCgesvdaStridedBatched(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    int rank,
    int m,
    int n,
    const mcComplex *A,
    int lda,
    long long int strideA,
    float *S,
    long long int strideS,
    mcComplex *U,
    int ldu,
    long long int strideU,
    mcComplex *V,
    int ldv,
    long long int strideV,
```

(下页继续)

(续上页)

```
    mcComplex *work,
    int lwork,
    int *info,
    double *h_R_nrmF,
    int batchSize);

mcsolverStatus_t
mcsolverDnZgesvdaStridedBatched(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    int rank,
    int m,
    int n,
    const mcDoubleComplex *A,
    int lda,
    long long int strideA,
    double *S,
    long long int strideS,
    mcDoubleComplex *U,
    int ldu,
    long long int strideU,
    mcDoubleComplex *V,
    int ldv,
    long long int strideV,
    mcDoubleComplex *work,
    int lwork,
    int *info,
    double *h_R_nrmF,
    int batchSize);
```

该函数 gesvda （a 代表近似）用于近似计算一个 m×n 矩阵 A 的奇异值分解以及对应的左奇异向量和右奇异向量。经济形式的奇异值分解可以表示为：

$$A = U * \Sigma * V^H$$

其中 Σ 是一个 m×n 的矩阵，U 是一个 m×m 的酉矩阵，同时 V 是一个 n×n 的酉矩阵。Σ 的对角元素是矩阵 A 的奇异值：它们是实数且非负，并按降序返回。U 和 V 是矩阵 A 的左奇异向量和右奇异向量。如果奇异值取值距零较远，那么左奇异向量 U 就是准确的。换句话说，奇异值和左奇异向量的准确性取决于奇异值与零之间的距离。

gesvda 是一个计算 A**T*A 的特征值以近似取得奇异值和奇异向量的函数。它生成矩阵 U 和 V ，并将矩阵 A 转换为以下形式：

$$U^H * A * V = S + E$$

其中 S 是一个对角矩阵，而 E 则取决于舍入误差。在某些条件下，U ，V 和 S 可以近似于单精度浮点数的机器零。一般来说，V 是酉矩阵，同时 S 比 U 更准确。

输入参数 rank 决定了在参数 S ，U 和 V 中计算的奇异值和奇异向量的数量。

输出参数 h_RnrmF 用于计算残差的 Frobenius 范数。

$$A - U * S * V^H$$

除非参数 rank 等于 n 。否则，h_RnrmF 将报告。

$$||U * S * V^H|| = ||S||$$

在 Frobenius 范数的定义内，即表明，U 与单位矩阵距离多远。

gesvdaStridedBatched 函数对每个矩阵执行 gesvda 操作。它要求所有矩阵的大小都同为 m,n 且以连续的方式打包。

每个矩阵都以列为主，且具有前导维度 lda ，因此随机访问的公式是。类似地，矩阵 S 的随机访问公式是，矩阵 U 的随机访问公式是，矩阵 V 的随机访问公式是。

用户需要提供一个工作空间，该工作空间由输入参数 work 指定。输入参数 lwork 表示工作空间的大小，并且可以通过 gesvdaStridedBatched_bufferSize() 函数返回。

输出参数 info 是一个大小为 batchSize 的整数数组。如果函数返回 MCSOLVER_STATUS_INVALID_VALUE ，则第一个元素 info[0] = -i （小于零）表示第 i 个参数错误（不包括句柄）。否则，如果 info[i] = min(m,n)+1 ，则 gesvdaStridedBatched 在给定的容差下不收敛于第 i 个矩阵。

备注 1：该例程返回的是 V ，而不是 V^H 。这与 gesvd 不同。

如果用户对奇异值和奇异向量的准确性有信心，例如满足某些条件（所需奇异值取值离零较远），那么通过将空指针传递给 h_RnrmF ，即不计算残差范数，就可以提高性能。

API of gesvda

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库的句柄。
jobz	host	input	该参数用于指定计算奇异值还是同时计算奇异值和奇异向量: jobz = MCSOLVER_EIG_MODE_NOVECTOR：表示仅计算奇异值; jobz = MCSOLVER_EIG_MODE_VECTOR：表示同时计算奇异值和奇异向量。
rank	host	input	奇异值的数量（从大到小排列）。
m	host	input	矩阵 Aj 的行数。
n	host	input	矩阵 Aj 的列数。
A	device	input	维度为 strideA *batchSize 的 <type> 数组，其中 lda 不小于 max(1,m) 。矩阵 Aj 的维度为 m *n 。
lda	host	input	用于存储矩阵 Aj 的二维数组的前导维度。
strideA	host	input	类型为双长整型的值，其表示了 A[i] 和 A[i+1] 之间的地址偏移量。同时，strideA 的值不小于 lda*n 。
S	device	output	维度为 strideS*batchSize 的实数数组，用于按降序存储矩阵 Aj 的奇异值。矩阵 Sj 的维度为 rank *1 。
strideS	host	input	类型为双长整型的值，该值表示 S[i] 和 S[i+1] 之间的地址偏移量。strideS 不小于 rank 。
U	device	output	维度为 strideU *batchSize 的 <type> 数组。Uj 包含了 Aj 的左奇异向量。Uj 的维度为 m *rank 。
ldu	host	input	用于存储矩阵 Uj 的二维数组的前导维度。ldu 不小于 max(1, m) 。
strideU	host	input	类型为双长整型的值，该值表示 U[i] 和 U[i+1] 之间的地址偏移量。strideU 不小于 ldu*rank 。
V	device	output	维度为 strideV *batchSize 的 <type> 数组。Vj 包含了 Aj 的右奇异向量。Vj 的维度为 n *rank 。
ldv	host	input	用于存储矩阵 Vj 的二维数组的前导维度。ldv 不小于 max(1, n) 。
strideV	host	input	类型为双长整型的值，该值表示 V[i] 和 V[i+1] 之间的地址偏移量。strideV 不小于 ldv*rank 。
work	device	in/out	用作工作空间大小为 lwork 的 <type> 数组。
lwork	host	input	gesvdaStridedBatched_bufferSize 函数返回的 work 大小。

下页继续

表 2.22 – 续上页

参数	内存	In/out	含义
info	device	output	一个维度为 batchSize 的整数数组。如果返回 MCSOLVER_STATUS_INVALID_VALUE，则 info[0] = -i（小于零）表示第 i 个参数错误（不包括句柄）。否则，如果 info[i] = 0，则表示操作成功。如果 info[i] = min(m,n)+1，gesvdaStridedBatched 无法收敛于第 i 个矩阵。
h_RnrmF	host	output	大小为 batchSize 的 <double> 数组。h_RnrmF[i] 表示第 i 个矩阵的残差范数。
batchSize	host	input	矩阵的数量。batchSize 不小于 1。

返回状态

MCSOLVER_STATUS_SUCCESS	操作已成功完成。
MCSOLVER_STATUS_NOT_INITIALIZED	库未初始化。
MCSOLVER_STATUS_INVALID_VALUE	传递了无效的参数（m,n<0 或 lda<max(1,m) 或 ldu<max(1,m) 或 ldv<max(1,n) 或 strideA<lda*n 或 strideS<rank 或 strideU<ldu*rank 或 strideV<ldv*rank 或 batchSize<1 或者 jobz 不是 MCSOLVER_EIG_MODE_NOVECTOR 或 MCSOLVER_EIG_MODE_VECTOR）。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.3.11 mcsolverDn<t>syevd()

这些辅助函数计算所需的工作缓冲区的大小。

```
mcsolverStatus_t
mcsolverDnSsyevd_bufferSize(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    mcblasFillMode_t uplo,
    int n,
    const float *A,
    int lda,
    const float *W,
    int *lwork);

mcsolverStatus_t
mcsolverDnDsyevd_bufferSize(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    mcblasFillMode_t uplo,
    int n,
    const double *A,
    int lda,
    const double *W,
    int *lwork);

mcsolverStatus_t
mcsolverDnCheevd_bufferSize(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
```

(下页继续)

(续上页)

```
    mcblasFillMode_t uplo,
    int n,
    const mcComplex *A,
    int lda,
    const float *W,
    int *lwork);

mcsolverStatus_t
mcsolverDnZheevd_bufferSize(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    mcblasFillMode_t uplo,
    int n,
    const mcDoubleComplex *A,
    int lda,
    const double *W,
    int *lwork);
```

S 和 D 数据类型分别表示单精度实数和双精度实数。

```
mcsolverStatus_t
mcsolverDnSsyevd(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    mcblasFillMode_t uplo,
    int n,
    float *A,
    int lda,
    float *W,
    float *work,
    int lwork,
    int *devInfo);

mcsolverStatus_t
mcsolverDnDsyevd(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    mcblasFillMode_t uplo,
    int n,
    double *A,
    int lda,
    double *W,
    double *work,
    int lwork,
    int *devInfo);
```

C 和 Z 数据类型分别表示单精度复数和双精度复数。

```
mcsolverStatus_t
mcsolverDnCheevd(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    mcblasFillMode_t uplo,
    int n,
    mcComplex *A,
    int lda,
    float *W,
```

(下页继续)

(续上页)

```
    mcComplex *work,
    int lwork,
    int *devInfo);

mcsolverStatus_t
mcsolverDnZheevd(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    mcblasFillMode_t uplo,
    int n,
    mcDoubleComplex *A,
    int lda,
    double *W,
    mcDoubleComplex *work,
    int lwork,
    int *devInfo);
```

这个函数计算 n×n 对称 (厄米特) 矩阵 A 的特征值和特征向量。标准对称特征值问题如下：

$$A * V = V * \Lambda$$

其中 Λ 是一个 n×n 实数对角矩阵。V 是一个 n×n 酉矩阵。Λ 的对角元素是 A 的特征值，按升序排列。

用户必须提供输入参数 work 所指向的工作空间。输入参数 lwork 表示工作空间的大小，并且由 syevd_bufferSize() 返回。

如果输出参数 devInfo = -i (小于 0)，则第 i 个参数是错误的 (不包含句柄)。如果 devInfo = i (大于 0)，中间三对角形式的 i 个非对角元素没有收敛于零。

如果 jobz =mcsolver_EIG_MODE_VECTOR，则 A 包含矩阵 A 的正交特征向量。特征向量通过分治算法计算。

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库的句柄。
jobz	host	input	该参数用于指定计算特征值或计算特征值和特征向量对: jobz = MCSOLVER_EIG_MODE_NOVECTOR：仅计算特征值; jobz = MCSOLVER_EIG_MODE_VECTOR：计算特征值和特征向量。
uplo	host	input	说明存储了 A 矩阵的哪个部分。uplo = MCBLAS_FILL_MODE_LOWER：存储了 A 矩阵的下三角部分。uplo = MCBLAS_FILL_MODE_UPPER: 存储了 A 矩阵的上三角部分。
n	host	input	矩阵 Aj 的行（或列）数。
A	device	in/out	<type> 数组的维度为 lda * n，其中 lda 不小于 max(1, n)。如果 uplo = MCBLAS_FILL_MODE_UPPER，则 A 的前 n-by-n 的上三角部分包含了矩阵 A 的上三角部分。如果 uplo = MCBLAS_FILL_MODE_LOWER，则 A 的前 n-by-n 的下三角部分包含了矩阵 A 的下三角部分。在计算完成后，如果 jobz = MCSOLVER_EIG_MODE_VECTOR，且 devInfo = 0，A 将包含矩阵 A 的正交特征向量。如果 jobz = MCSOLVER_EIG_MODE_NOVECTOR，则 A 的内容将被清空。
lda	host	input	用于存储矩阵 A 的二维数组的前导维度。
W	device	output	维度为 n 的实数数组。该数组存储了矩阵 A 的特征值并按升序排列，即满足 W(i) <= W(i+1)。
work	device	in/out	用作工作空间大小为 lwork 的 <type> 数组。
Lwork	host	input	syevd_bufferSize 函数返回的 work 大小。
devInfo	device	output	如果 devInfo = 0，表示操作成功。如果 devInfo = -i，表示第 i 个参数错误 (不包括句柄)。如果 devInfo = i (>0)，则 devInfo 表示中间三对角形式中的 i 个非对角元素未收敛至零。

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。
MCSOLVER_STATUS_NOT_INITIALIZED	库未进行初始化
MCSOLVER_STATUS_INVALID_VALUE	传递的参数无效 (n<0 或 lda<max(1,n))。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.3.12 mcsolverDnSyevd()

下面的辅助函数可以计算所需的预分配缓冲区大小。

```
mcsolverStatus_t
    mcsolverDnSyevd_bufferSize(
    mcsolverDnHandle_t handle,
    mcsolverParams_t params,
    mcsolverEigMode_t jobz,
    mcblasFillMode_t uplo,
    int n,
    macaDataType dataTypeA,
    const void *A,
    int64_t lda,
    macaDataType dataTypeW,
    const void *W,
    macaDataType computeType,
    size_t *workspaceInBytes);
```

下面的例程

```
mcsolverStatus_t
    mcsolverDnSyevd(
    mcsolverDnHandle_t handle,
    mcsolverParams_t params,
    mcsolverEigMode_t jobz,
    mcblasFillMode_t uplo,
    int n,
    macaDataType dataTypeA,
    const void *A,
    int64_t lda,
    macaDataType dataTypeW,
    const void *W,
    macaDataType computeType,
    void *pBuffer,
    size_t workspaceInBytes,
    int *info);
```

借助通用 API 接口，计算 n×n 对称 (厄米特) 矩阵 A 的特征值和特征向量。标准对称特征值问题如下

$$A * V = V * \Lambda$$

其中 Λ 是一个 n×n 实数对角矩阵。V 是一个 n×n 酉矩阵。Λ 的对角线元素是 A 的特征值，按升序排列。

用户需要提供由输入参数 pBuffer 指向的工作空间。输入参数 workspaceInBytes 是工作空间的字节大小，并且由 mcsolverDnSyevd_bufferSize() 返回。

如果输出参数 info=-i (小于 0)，则第 i 个参数是错误的 (不包含句柄)。如果 info=i (大于 0)，中间三对角形式的 i 个非对角元素不收敛于零。

如果 jobz = mcsolver_EIG_MODE_VECTOR,，则 A 包含矩阵 A 的正交特征向量。特征向量通过分治算法计算。

目前，mcsolverDnSyevd 仅支持默认的算法。

mcsolverDnSyevd 支持的算法表

mcsolver_ALG_0 or NULL	默认算法

API of mcsolverDnSyevd

参数	内存	in/out	含义
handle	host	input	mcsolverDn 库上下文的句柄。
params	host	input	由 mcsolverDnSetAdvOptions 收集的信息的结构体。
jobz	host	input	指定计算仅特征值或计算特征值和特征向量的选项; jobz = MCSOLVER_EIG_MODE_NOVECTOR：仅计算特征值; jobz = MCSOLVER_EIG_MODE_VECTOR：计算特征值和特征向量。
uplo	host	input	指定 A 的哪个部分被存储。uplo = MCBLAS_FILL_MODE_LOWER：存储 A 的下三角部分。uplo = MCBLAS_FILL_MODE_UPPER：存储 A 的上三角部分。
n	host	input	矩阵 A 的行数（或列数）。
dataTypeA	host	in	数组 A 的数据类型。
A	device	in/out	维度为 lda * n 的数组，其中 lda 不小于 max(1, n)。如果 uplo = MCBLAS_FILL_MODE_UPPER，则 A 的前 n * n 的上三角部分包含矩阵 A 的上三角部分。如果 uplo = MCBLAS_FILL_MODE_LOWER，则 A 的前 n * n 的下三角部分包含矩阵 A 的下三角部分。在退出时，如果 jobz = MCSOLVER_EIG_MODE_VECTOR，且 info = 0，则 A 包含矩阵 A 的正交特征向量。如果 jobz = MCSOLVER_EIG_MODE_NOVECTOR，则 A 的内容被清空。
lda	host	input	用于存储矩阵 A 的二维数组的前导维度。
dataTypeW	host	in	数组 W 的数据类型。
W	device	output	长度为 n 的实数数组。矩阵 A 的特征值，按升序排列，即排序为 W(i) <= W(i+1)。
computeType	host	in	计算的数据类型。
pBuffer	device	in/out	工作空间。大小为 workspaceInBytes 字节的 void 类型数组。
workspaceInBytes	host	input	pBuffer 的字节大小，由 mcsolverDnSyevd_bufferSize 返回。
info	device	output	如果 info = 0，操作成功。如果 info = -i，第 i 个参数错误（不包括句柄）。如果 info = i(>0)，info 指示第 i 个中间三对角形式的非对角元素未收敛为零。

通用 API 接口有三种不同类型，dataTypeA 是矩阵 A 的数据类型。dataTypeW 是矩阵 W 的数据类型以及 computeType 是操作的计算类型。mcsolverDnSyevd 仅支持以下四种组合。

数据类型和计算类型的有效组合

DataTypeA	DataTypeW	ComputeType	Meaning
MACA_R_32F	MACA_R_32F	MACA_R_32F	SSYEVD
MACA_R_64F	MACA_R_64F	MACA_R_64F	DSYEVD
MACA_C_32F	MACA_R_32F	MACA_C_32F	CHEEVD
MACA_C_64F	MACA_R_64F	MACA_C_64F	ZHEEVD

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。
MCSOLVER_STATUS_NOT_INITIALIZED	库未进行初始化。
MCSOLVER_STATUS_INVALID_VALUE	传递的参数无效 (n<0 或 lda<max(1,n))。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.3.13 mcsolverDn<t>syevdx()

下面的辅助函数可以计算需要预先分配的缓冲区的大小。

```
mcsolverStatus
mcsolverDnSsyevdx_bufferSize(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    mcsolverEigRange_t range,
    mcblasFillMode_t uplo,
    int n,
    const float *A,
    int lda,
    float vl,
    float vu,
    int il,
    int iu,
    int *h_meig,
    const float *W,
    int *lwork);

mcsolverStatus
mcsolverDnDsyevdx_bufferSize(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    mcsolverEigRange_t range,
    mcblasFillMode_t uplo,
    int n,
    const double *A,
    int lda,
    double vl,
    double vu,
    int il,
    int iu,
    int *h_meig,
    const double *W,
    int *lwork);

mcsolverStatus
mcsolverDnCheevdx_bufferSize(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    mcsolverEigRange_t range,
    mcblasFillMode_t uplo,
    int n,
    const mcComplex *A,
    int lda,
    float vl,
    float vu,
```

(下页继续)

(续上页)

```
    int il,
    int iu,
    int *h_meig,
    const float *W,
    int *lwork);

mcsolverStatus
mcsolverDnZheevdx_bufferSize(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    mcsolverEigRange_t range,
    mcblasFillMode_t uplo,
    int n,
    const mcDoubleComplex *A,
    int lda,
    double vl,
    double vu,
    int il,
    int iu,
    int *h_meig,
    const double *W,
    int *lwork);
```

S 和 D 数据类型分别是单精度和双精度实数。

```
mcsolverStatus
mcsolverDnSsyevdx(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    mcsolverEigRange_t range,
    mcblasFillMode_t uplo,
    int n,
    float *A,
    int lda,
    float vl,
    float vu,
    int il,
    int iu,
    int *h_meig,
    float *W,
    float *work,
    int lwork,
    int *devInfo);

mcsolverStatus
mcsolverDnDsyevdx(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    mcsolverEigRange_t range,
    mcblasFillMode_t uplo,
    int n,
    double *A,
    int lda,
    double vl,
    double vu,
    int il,
```

(下页继续)

(续上页)

```
    int iu,
    int *h_meig,
    double *W,
    double *work,
    int lwork,
    int *devInfo);
```

C 和 Z 数据类型分别是单精度和双精度复数。

```
mcsolverStatus
mcsolverDnCheevdx(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    mcsolverEigRange_t range,
    mcblasFillMode_t uplo,
    int n,
    mcComplex *A,
    int lda,
    float vl,
    float vu,
    int il,
    int iu,
    int *h_meig,
    float *W,
    mcComplex *work,
    int lwork,
    int *devInfo);

mcsolverStatus
mcsolverDnZheevdx(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    mcsolverEigRange_t range,
    mcblasFillMode_t uplo,
    int n,
    mcDoubleComplex *A,
    int lda,
    double vl,
    double vu,
    int il,
    int iu,
    int *h_meig,
    double *W,
    mcDoubleComplex *work,
    int lwork,
    int *devInfo);
```

此函数计算对称（厄米特）的 n*n 矩阵 A 的所有或选定的特征值，并可选择计算特征向量。标准的对称特征值问题为：

$$A * V = V * \Lambda$$

其中，Λ 是一个实数的 n×h_meig 对角矩阵，V 是一个 n×h_meig 的酉矩阵。h_meig 是例程计算的特征值/特征向量的数量，当需要计算整个谱（例如，range = MCSOLVER_EIG_RANGE_ALL）时，h_meig 等于 n。Λ 的对角元素是按升序排列的 A 的特征值。

用户必须提供由输入参数 work 指向的工作空间。输入参数 lwork 表示工作空间的大小，它由 syevdx_bufferSize() 返回。

如果输出参数 devInfo = -i （小于零），则第 i 个参数错误（不包括句柄）。如果 devInfo = i （大于零），则中间三对角形式的 i 个非对角元素未收敛于零。

如果 jobz = MCSOLVER_EIG_MODE_VECTOR ，则 A 包含矩阵 A 的正交特征向量。特征向量是通过分治算法计算得出的。

API of syevdx

参数	内存	in/out	含义
handle	host	input	mcsolverDN 库上下文的句柄。
jobz	host	input	指定选项，用于计算特征值或计算特征对： • jobz = MCSOLVER_EIG_MODE_NOVECTOR： 仅计算特征值。 • jobz = MCSOLVER_EIG_MODE_VECTOR： 计算特征值和特征向量。
range	host	input	指定选项，用于计算需要计算的特征值和可选的特征向量。 • range = MCSOLVER_EIG_RANGE_ALL：找到所有特征值/特征向量，将成为经典的 syevd/heevd 例程。 • range = MCSOLVER_EIG_RANGE_V： 找到半开区间 (vl,vu] 内的所有特征值/特征向量。 • range = MCSOLVER_EIG_RANGE_I： 找到第 il 到第 iu 个特征值/特征向量。
uplo	host	input	指定存储在 A 的哪个部分。 • uplo = MCBLAS_FILL_MODE_LOWER： 存储 A 的下三角部分。 • uplo = MCBLAS_FILL_MODE_UPPER： 存储 A 的上三角部分。。
n	host	input	矩阵 A 的行数（或列数）。
A	device	in/out	维度为 lda * n 的 <type> 数组，其中 lda 不小于 max(1,n) 。 • 如果 uplo = MCBLAS_FILL_MODE_UPPER ，则 A 的前 n * n 的上三角部分包含矩阵 A 的上三角部分。 • 如果 uplo = MCBLAS_FILL_MODE_LOWER ，则 A 的前 n * n 的下三角部分包含矩阵 A 的下三角部分。 • 在退出时，如果 jobz = MCSOLVER_EIG_MODE_VECTOR ，且 devInfo = 0 ，则 A 包含矩阵 A 的正交特征向量。 • 如果 jobz = MCSOLVER_EIG_MODE_NOVECTOR ，则 A 的内容被清空。
lda	host	input	用于存储矩阵 A 的二维数组的前导维度。lda 不小于 max(1,n) 。
vl,vu	host	input	实数值浮点或双精度 (对于 (C,S) 或 (Z ,D) 精度)。 • 如果 range = MCSOLVER_EIG_RANGE_V ，则特征值需在此半开区间 (vl ,vu] 内搜索。vl > vu。 • 如 果 range = MCSOLVER_EIG_RANGE_ALL 或 range = MCSOLVER_EIG_RANGE_I ，则不引用。 请注意，如果特征值非常接近，两个不同的特征值例程可能会在同一区间内找到稍微不同数量的特征值。这是因为不同的特征值算法，甚至是相同算法但不同运行，可能会在某些四舍五入误差接近机器精度的情况下找到特征值。因此，如果用户想要确保在区间边界内不会漏掉任何特征值，我们建议用户从区间边界减去/加上 epsilon(机器精度)，例如 (vl=vl-eps,vu=vu+eps] 。这个建议对于 mcSOLVER 或 LAPACK 中的任何选择例程都是有效的。

下页继续

表 2.25 – 续上页

参数	内存	in/out	含义
il,iu	host	input	整数。 • 如果 range = MCSOLVER_EIG_RANGE_I，则返回的特征值的最小和最大索引（按升序排列）。 • 如果 n > 0，则 1 <= il <= iu <= n。 • 如果 n = 0，则 il = 1 和 iu = 0。 • 如果 range = MCSOLVER_EIG_RANGE_ALL 或 range = MCSOLVER_EIG_RANGE_V，则不引用。
h_meig	host	output	整数。找到的特征值总数。0 <= h_meig <= n。 • 如果 range = MCSOLVER_EIG_RANGE_ALL，则 h_meig = n。 • 如果 range = MCSOLVER_EIG_RANGE_I，则 h_meig = iu-il+1。
W	device	output	维度为 n 的实数数组。矩阵 A 的特征值，按升序排列，即排序为 W(i) <= W(i+1)。
work	device	in/out	工作空间，大小为 lwork 的 <type> 数组。
lwork	host	input	由 syevdx_bufferSize 返回的 work 的大小。
devInfo	device	output	• 如果 devInfo = 0，则操作成功。 • 如果 devInfo = -i，则第 i 个参数错误（不包括句柄）。 • 如果 devInfo = i (大于 0)，则 devInfo 指示中间三对角形式的 i 个非对角元素未收敛于零。

返回状态

MCSOLVER_STATUS_SUCCESS	操作成功完成。
MCSOLVER_STATUS_NOT_INITIALIZED	库未初始化。
MCSOLVER_STATUS_INVALID_VALUE	传递了无效的参数（n<0、lda<max(1,n)、jobz 不是 MCSOLVER_EIG_MODE_NOVECTOR 或 MCSOLVER_EIG_MODE_VECTOR、range 不是 MCSOLVER_EIG_RANGE_ALL 或 MCSOLVER_EIG_RANGE_V 或 MCSOLVER_EIG_RANGE_I，或 uplo 不是 MCBLAS_FILL_MODE_LOWER 或 MCBLAS_FILL_MODE_UPPER）。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.3.14 mcsolverDnSyevdx()

下面的辅助函数可以计算预分配缓冲区所需的大小。

```
mcsolverStatus_t
     mcsolverDnSyevdx_bufferSize(
     mcsolverDnHandle_t handle,
     mcsolverParams_t params,
     mcsolverEigMode_t jobz,
     mcsolverEigRange_t range,
     mcblasFillMode_t uplo,
     int n,
     macaDataType dataTypeA,
     const void *A,
```

(下页继续)

(续上页)

```
    int64_t lda,
    void *vl,
    void *vu,
    int64_t il,
    int64_t iu,
    int64_t *h_meig,
    macaDataType dataTypeW,
    const void *W,
    macaDataType computeType,
    size_t *workspaceInBytes);
```

以下例程：

```
mcsolverStatus_t
    mcsolverDnSyevdx (
    mcsolverDnHandle_t handle,
    mcsolverParams_t params,
    mcsolverEigMode_t jobz,
    mcsolverEigRange_t range,
    mcblasFillMode_t uplo,
    int n,
    macaDataType dataTypeA,
    const void *A,
    int64_t lda,
    void *vl,
    void *vu,
    int64_t il,
    int64_t iu,
    int64_t *h_meig,
    macaDataType dataTypeW,
    const void *W,
    macaDataType computeType,
    void *pBuffer,
    size_t workspaceInBytes,
    int *info);
```

使用通用 API 接口计算对称（厄米特）的 n×n 矩阵 A 的所有或选定的特征值，并可选择计算特征向量。标准的对称特征值问题为：

$$A * V = V * \Lambda$$

其中，Λ 是一个实数的 n×h_meig 对角矩阵，V 是一个 n×h_meig 的酉矩阵。h_meig 是例程计算的特征值/特征向量的数量，当需要计算整个谱（例如，range = MCSOLVER_EIG_RANGE_ALL）时，h_meig 等于 n。Λ 的对角元素是按升序排列的 A 的特征值。

用户必须提供由输入参数 pBuffer 指向的工作空间。输入参数 workspaceInBytes 表示工作空间的大小（以字节为单位），由 mcsolverDnSyevdx_bufferSize() 返回。

如果输出参数 info=-i （小于零），则第 i 个参数错误（不包括句柄）。如果 info=i （大于零），则中间三对角形式的 i 个非对角元素未收敛于零。

如果 jobz = MCSOLVER_EIG_MODE_VECTOR ，则 A 包含矩阵 A 的正交特征向量。特征向量是通过分治算法计算得出的。

目前，mcsolverDnSyevdx 仅支持默认算法。

mcsolverDnSyevdx 支持的算法表

MCSOLVER_ALG_0 or NULL	默认算法

mcsolverDnSyevdx_bufferSize 和 mcsolverDnSyevdx 的输入参数列表：

API of mcsolverDnSyevdx

参数	内存	in/out	含义
handle	host	input	mcsolverDN 库上下文的句柄。
params	host	input	由 mcsolverDnSetAdvOptions 收集的信息的结构体。
jobz	host	input	指定选项，要么仅计算特征值，要么计算特征值和特征向量：jobz = MCSOLVER_EIG_MODE_NOVECTOR：仅计算特征值；jobz = MCSOLVER_EIG_MODE_VECTOR：计算特征值和特征向量。
range	host	input	指定选项，需要计算哪些特征值和可选择的特征向量：range = MCSOLVER_EIG_RANGE_ALL：将找到所有特征值/特征向量，将变为经典的 syevd/heevd 例程；range = MCSOLVER_EIG_RANGE_V：在半开区间（vl,vu] 中找到所有特征值/特征向量；range = MCSOLVER_EIG_RANGE_I：找到 il 到 iu 的特征值/特征向量；
uplo	host	input	指定存储在 A 的哪个部分。uplo = MCBLAS_FILL_MODE_LOWER：存储 A 的下三角部分。uplo = MCBLAS_FILL_MODE_UPPER：存储 A 的上三角部分。
n	host	input	矩阵 A 的行数（或列数）。
dataTypeA	host	in	数组 A 的数据类型。
A	device	in/out	维度为 lda * n 的数组，其中 lda 不小于 max(1, n)。如果 uplo = MCBLAS_FILL_MODE_UPPER，则矩阵 A 的前 n * n 个上三角元素包含矩阵 A 的上三角部分。如果 uplo = MCBLAS_FILL_MODE_LOWER，则矩阵 A 的前 n * n 个下三角元素包含矩阵 A 的下三角部分。在退出时，如果 jobz = MCSOLVER_EIG_MODE_VECTOR，且 info = 0，矩阵 A 包含矩阵 A 的正交特征向量。如果 jobz = MCSOLVER_EIG_MODE_NOVECTOR，则矩阵 A 的内容被清空。
lda	host	input	用于存储矩阵 A 的二维数组的前导维度。lda 不小于 max(1,n)。
vl,vu	host	input	如果 range = MCSOLVER_EIG_RANGE_V，则是要搜索特征值的区间的下界和上界。vl > vu。如果 range = MCSOLVER_EIG_RANGE_ALL 或 range = MCSOLVER_EIG_RANGE_I，则不被引用。请注意，如果特征值非常接近，众所周知，两个不同的特征值例程可能会在同一区间内找到稍微不同数量的特征值。这是因为不同的特征值算法，甚至相同的算法但不同的运行，可能会在接近机器精度的舍入误差范围内找到特征值。因此，如果用户希望确保不会错过区间边界内的任何特征值，建议用户从区间边界中减去/添加 epsilon（机器精度），例如（vl=vl-eps ,vu=vu+eps）。这个建议对于 mcSOLVER 或 LAPACK 中的任何选择性例程都是有效的。
il,iu	host	input	整数。如果 range = MCSOLVER_EIG_RANGE_I，则是要返回的最小和最大特征值的索引（按升序排列）。如果 n > 0，则 1 <= il <= iu <= n；如果 n = 0，则 il = 1,iu = 0。如果 range = MCSOLVER_EIG_RANGE_ALL 或 range = MCSOLVER_EIG_RANGE_V，则不被引用。

下页继续

表 2.26 – 续上页

h_meig	host	output	整数。找到的特征值的总数。0 <= h_meig <= n。如果 range = MCSOLVER_EIG_RANGE_ALL，则 h_meig = n；如果 range = MCSOLVER_EIG_RANGE_I，则 h_meig = iu - il + 1。
dataTypeW	host	in	数组 W 的数据类型。
W	device	output	一个维度为 n 的实数组。矩阵 A 的特征值，按升序排列，即排序使得 W(i) <= W(i+1)。
computeType	host	in	计算的数据类型。
pBuffer	device	in/ out	工作空间。类型为 void 的数组，大小为 workspaceInBytes 字节。
workspaceInBytes	host	input	pBuffer 的字节数大小，由 mcsolverDnSyevdx_bufferSize 返回。
info	device	output	如果 info=0，则操作成功。如果 info=-i，则第 i 个参数错误（不包括句柄）。如果 info = i(>0)，则 info 指示中间三对角形式的 i 个非对角元素未收敛于零。

通用 API 有三种不同类型，dataTypeA 是矩阵 A 的数据类型，dataTypeW 是矩阵 W 的数据类型，computeType 是操作的计算类型。mcsolverDnSyevdx 仅支持以下四种组合。

数据类型和计算类型的有效组合

DataTypeA	DataTypeW	ComputeType	Meaning
MACA_R_32F	MACA_R_32F	MACA_R_32F	SSYEVDX
MACA_R_64F	MACA_R_64F	MACA_R_64F	DSYEVDX
MACA_C_32F	MACA_R_32F	MACA_C_32F	CHEEVDX
MACA_C_64F	MACA_R_64F	MACA_C_64F	ZHEEVDX

返回状态

MCSOLVER_STATUS_SUCCESS	操作成功完成。
MCSOLVER_STATUS_NOT_INITIALIZED	库未初始化。
MCSOLVER_STATUS_INVALID_VALUE	传递了无效的参数（n<0，或 lda<max(1,n)，或 jobz 不是 MCSOLVER_EIG_MODE_NOVECTOR 或 MCSOLVER_EIG_MODE_VECTOR，或 range 不是 MCSOLVER_EIG_RANGE_ALL 或 MCSOLVER_EIG_RANGE_V 或 MCSOLVER_EIG_RANGE_I，或 uplo 不是 MCBLAS_FILL_MODE_LOWER 或 MCBLAS_FILL_MODE_UPPER）。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.3.15 mcsolverDn<t>sygvd()

下面的辅助函数可以计算所需的预分配缓冲区大小。

```
mcsolverStatus_t
mcsolverDnSsygvd_bufferSize(
    mcsolverDnHandle_t handle,
    mcsolverEigType_t itype,
    mcsolverEigMode_t jobz,
    mcblasFillMode_t uplo,
    int n,
    const float *A,
```

(下页继续)

(续上页)

```
    int lda,
    const float *B,
    int ldb,
    const float *W,
    int *lwork);

mcsolverStatus_t
mcsolverDnDsygvd_bufferSize(
    mcsolverDnHandle_t handle,
    mcsolverEigType_t itype,
    mcsolverEigMode_t jobz,
    mcblasFillMode_t uplo,
    int n,
    const double *A,
    int lda,
    const double *B,
    int ldb,
    const double *W,
    int *lwork);

mcsolverStatus_t
mcsolverDnChegvd_bufferSize(
    mcsolverDnHandle_t handle,
    mcsolverEigType_t itype,
    mcsolverEigMode_t jobz,
    mcblasFillMode_t uplo,
    int n,
    const mcComplex *A,
    int lda,
    const mcComplex *B,
    int ldb,
    const float *W,
    int *lwork);

mcsolverStatus_t
mcsolverDnZhegvd_bufferSize(
    mcsolverDnHandle_t handle,
    mcsolverEigType_t itype,
    mcsolverEigMode_t jobz,
    mcblasFillMode_t uplo,
    int n,
    const mcDoubleComplex *A,
    int lda,
    const mcDoubleComplex *B,
    int ldb,
    const double *W,
    int *lwork);
```

S 和 D 数据类型分别表示单精度实数和双精度实数。

```
mcsolverStatus_t
mcsolverDnSsygvd(
    mcsolverDnHandle_t handle,
    mcsolverEigType_t itype,
    mcsolverEigMode_t jobz,
    mcblasFillMode_t uplo,
```

(下页继续)

(续上页)

```
    int n,
    float *A,
    int lda,
    float *B,
    int ldb,
    float *W,
    float *work,
    int lwork,
    int *devInfo);

mcsolverStatus_t
mcsolverDnDsygvd(
    mcsolverDnHandle_t handle,
    mcsolverEigType_t itype,
    mcsolverEigMode_t jobz,
    mcblasFillMode_t uplo,
    int n,
    double *A,
    int lda,
    double *B,
    int ldb,
    double *W,
    double *work,
    int lwork,
    int *devInfo);
```

C 和 Z 数据类型分别表示单精度复数和双精度复数。

```
mcsolverStatus_t
mcsolverDnChegvd(
    mcsolverDnHandle_t handle,
    mcsolverEigType_t itype,
    mcsolverEigMode_t jobz,
    mcblasFillMode_t uplo,
    int n,
    mcComplex *A,
    int lda,
    mcComplex *B,
    int ldb,
    float *W,
    mcComplex *work,
    int lwork,
    int *devInfo);

mcsolverStatus_t
mcsolverDnZhegvd(
    mcsolverDnHandle_t handle,
    mcsolverEigType_t itype,
    mcsolverEigMode_t jobz,
    mcblasFillMode_t uplo,
    int n,
    mcDoubleComplex *A,
    int lda,
    mcDoubleComplex *B,
    int ldb,
    double *W,
```

(下页继续)

(续上页)

```
    mcDoubleComplex *work,
    int lwork,
    int *devInfo);
```

这个函数计算 n×n 对称 (厄米特) 矩阵对 (A,B) 的特征值和特征向量。广义对称定特征值问题是

用户必须提供输入参数 work 所指向的工作空间。输入参数 lwork 表示工作空间的大小，并且由 sygvd_bufferSize() 返回。

如果输出参数 devInfo=-i (小于 0)，则第 i 个参数是错误的 (不包含句柄)。如果 devInfo=i (i > 0 and i<=n) 且 jobz = mcsolver_EIG_MODE_NOVECTOR，则中间三对角形式的 i 个非对角元素不收敛于零。如果 devInfo=N + i (i > 0)，那么 B 的第 i 阶导数是不正定的。无法完成 B 的因式分解，也无法计算特征值或特征向量。

如果 jobz = mcsolver_EIG_MODE_VECTOR，A 包含矩阵 A 的正交特征向量。采用分治算法计算特征向量。

API of sygvd

参数	内存	in/out	含义
handle	host	input	mcsolverDN 库上下文的句柄。
itype	host	input	指定要解决的问题类型： • itype = MCSOLVER_EIG_TYPE_1: A*x = (lambda)*B*x. • itype = MCSOLVER_EIG_TYPE_2: A*B*x = (lambda)*x. • itype = MCSOLVER_EIG_TYPE_3: B*A*x = (lambda)*x.
jobz	host	input	指定选项，用于仅计算特征值或计算特征值和特征向量。 • jobz = MCSOLVER_EIG_MODE_NOVECTOR： 仅计算特征值。 • jobz = MCSOLVER_EIG_MODE_VECTOR： 计算特征值和特征向量。
uplo	host	input	指定存储在 A 和 B 中哪个部分。uplo = MCBLAS_FILL_MODE_LOWER： 存储 A 和 B 的下三角部分。uplo = MCBLAS_FILL_MODE_UPPER： 存储 A 和 B 的上三角部分。
n	host	input	矩阵 A 和 B 的行数（或列数)。
A	device	in/out	维度为 lda * n 的 <type> 数组，其中 lda 至少不小于 max(1, n) 。如果 uplo = MCBLAS_FILL_MODE_UPPER ，则矩阵 A 的前 n 行和前 n 列的上三角部分包含了矩阵 A 的上三角部分。如果 uplo = MCBLAS_FILL_MODE_LOWERV ，则矩阵 A 的前 n 行和前 n 列的下三角部分包含了矩阵 A 的下三角部分。在退出时，如果 jobz = MCSOLVER_EIG_MODE_VECTOR ，并且 devInfo = 0 ，则 A 包含矩阵 A 的正交特征向量。如果 jobz = MCSOLVER_EIG_MODE_NOVECTOR ，则 A 的内容被清空。
lda	host	input	用于存储矩阵 A 的二维数组的前导维度。lda 至少不小于 max(1, n) 。
B	device	in/out	维度为 ldb * n 的 <type> 数组。如果 uplo = MCBLAS_FILL_MODE_UPPER ，则矩阵 B 的前 n 行和前 n 列的上三角部分包含了矩阵 B 的上三角部分。如果 uplo = MCBLAS_FILL_MODE_LOWER ，则矩阵 B 的前 n 行和前 n 列的下三角部分包含了矩阵 B 的下三角部分。在退出时，如果 devInfo 小于 n ，则 B 被 Cholesky 分解的三角因子 U 或 L 覆盖。
ldb	host	input	用于存储矩阵 B 的二维数组的前导维度。ldb 至少不小于 max(1, n) 。
W	device	output	一个维度为 n 的实数数组。矩阵 A 的特征值，使 W(i) >= W(i+1) 。
work	device	in/out	工作空间，大小为 lwork 的 <type> 数组。

下页继续

表 2.27 – 续上页

参数	内存	in/out	含义
Lwork	host	input	由 sygvd_bufferSize 返回的 work 的大小。
devInfo	device	output	如果 devInfo = 0，操作成功。如果 devInfo = -i，则第 i 个参数是错误的（不包括句柄）。如果 devInfo = i(>0)，则 devInfo 指示 potrf 或 syevd 是错误的。

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。
MCSOLVER_STATUS_NOT_INITIALIZED	库未进行初始化。
MCSOLVER_STATUS_INVALID_VALUE	传递的参数无效 (n<0 或 lda<max(1,n))。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.3.16 mcsolverDn<t>sygvdx()

以下是可以计算预分配缓冲区所需大小的辅助函数：

```
mcsolverStatus_t
 mcsolverDnSsygvdx_bufferSize(
     mcsolverDnHandle_t handle,
     mcsolverEigType_t itype,
     mcsolverEigMode_t jobz,
     mcsolverEigRange_t range,
     mcblasFillMode_t uplo,
     int n,
     const float *A,
     int lda,
     const float *B,
     int ldb,
     float vl,
     float vu,
     int il,
     int iu,
     int *h_meig,
     const float *W,
     int *lwork);

mcsolverStatus_t
 mcsolverDnDsygvdx_bufferSize(
     mcsolverDnHandle_t handle,
     mcsolverEigType_t itype,
     mcsolverEigMode_t jobz,
     mcsolverEigRange_t range,
     mcblasFillMode_t uplo,
     int n,
     const double *A,
     int lda,
     const double *B,
     int ldb,
     double vl,
     double vu,
     int il,
     int iu,
     int *h_meig,
     const double *W,
```

(下页继续)

(续上页)

```
     int *lwork);

mcsolverStatus_t
 mcsolverDnChegvdx_bufferSize(
     mcsolverDnHandle_t handle,
     mcsolverEigType_t itype,
     mcsolverEigMode_t jobz,
     mcsolverEigRange_t range,
     mcblasFillMode_t uplo,
     int n,
     const mcComplex *A,
     int lda,
     const mcComplex *B,
     int ldb,
     float vl,
     float vu,
     int il,
     int iu,
     int *h_meig,
     const float *W,
     int *lwork);

mcsolverStatus_t
 mcsolverDnZhegvdx_bufferSize(
     mcsolverDnHandle_t handle,
     mcsolverEigType_t itype,
     mcsolverEigMode_t jobz,
     mcsolverEigRange_t range,
     mcblasFillMode_t uplo,
     int n,
     const mcDoubleComplex *A,
     int lda,
     const mcDoubleComplex *B,
     int ldb,
     double vl,
     double vu,
     int il,
     int iu,
     int *h_meig,
     const double *W,
     int *lwork);
```

S 和 D 数据类型分别表示单精度和双精度实数。

```
mcsolverStatus_t
 mcsolverDnSsygvdx(
     mcsolverDnHandle_t handle,
     mcsolverEigType_t itype,
     mcsolverEigMode_t jobz,
     mcsolverEigRange_t range,
     mcblasFillMode_t uplo,
     int n,
     float *A,
     int lda,
     float *B,
     int ldb,
```

(下页继续)

(续上页)

```
    float vl,
    float vu,
    int il,
    int iu,
    int *h_meig,
    float *W,
    float *work,
    int lwork,
    int *devInfo);

mcsolverStatus_t
 mcsolverDnDsygvdx(
    mcsolverDnHandle_t handle,
    mcsolverEigType_t itype,
    mcsolverEigMode_t jobz,
    mcsolverEigRange_t range,
    mcblasFillMode_t uplo,
    int n,
    double *A,
    int lda,
    double *B,
    int ldb,
    double vl,
    double vu,
    int il,
    int iu,
    int *h_meig,
    double *W,
    double *work,
    int lwork,
    int *devInfo);
```

C 和 Z 数据类型分别表示单精度和双精度复数。

```
mcsolverStatus_t
 mcsolverDnChegvdx(
    mcsolverDnHandle_t handle,
    mcsolverEigType_t itype,
    mcsolverEigMode_t jobz,
    mcsolverEigRange_t range,
    mcblasFillMode_t uplo,
    int n,
    mcComplex *A,
    int lda,
    mcComplex *B,
    int ldb,
    float vl,
    float vu,
    int il,
    int iu,
    int *h_meig,
    float *W,
    mcComplex *work,
    int lwork,
    int *devInfo);
```

(下页继续)

(续上页)

```
mcsolverStatus_t
 mcsolverDnZhegvdx(
     mcsolverDnHandle_t handle,
     mcsolverEigType_t itype,
     mcsolverEigMode_t jobz,
     mcsolverEigRange_t range,
     mcblasFillMode_t uplo,
     int n,
     mcDoubleComplex *A,
     int lda,
     mcDoubleComplex *B,
     int ldb,
     double vl,
     double vu,
     int il,
     int iu,
     int *h_meig,
     double *W,
     mcDoubleComplex *work,
     int lwork,
     int *devInfo);
```

该函数计算对称（厄米特）的矩阵 `n×n`（`A`,`B`）的所有或选定的特征值，可选地计算特征向量。广义对称正定特征值问题如下：

用户需要提供一个由输入参数 `work` 指向的工作空间。输入参数 `lwork` 表示工作空间的大小，由 `sygvdx_bufferSize()` 返回。

如果输出参数 `devInfo = -i`（小于零），则表示第 `i` 个参数错误（不计入句柄）。如果 `devInfo = i`（i 大于零且小于等于 n），且 `jobz` = MCSOLVER_EIG_MODE_NOVECTOR，表示中间三对角形式的 `i` 个非对角元素未收敛于零。如果 `devInfo = n + i` (i >0)，则 `B` 的前 `i` 阶次前导子矩阵不是正定的。无法完成 `B` 的分解，因此无法计算特征值或特征向量。

如果 `jobz` = MCSOLVER_EIG_MODE_VECTOR，矩阵 `A` 包含矩阵 `A` 的正交特征向量。特征向量由分治算法计算得出。

API of sygvdx

参数	内存	in/out	含义
handle	host	input	mcsolverDN 库上下文的句柄。
itype	host	input	指定要解决的问题类型： • itype = MCSOLVER_EIG_TYPE_1: A*x = (lambda)*B*x • itype = MCSOLVER_EIG_TYPE_2: A*B*x = (lambda)*x • itype = MCSOLVER_EIG_TYPE_3: B*A*x = (lambda)*x
jobz	host	input	指定选项以仅计算特征值或计算特征值和特征向量：jobz = MCSOLVER_EIG_MODE_NOVECTOR：仅计算特征值；jobz = MCSOLVER_EIG_MODE_VECTOR：计算特征值和特征向量。
range	host	input	指定需要计算的特征值和可选的特征向量的选择选项：range = MCSOLVER_EIG_RANGE_ALL：将找到所有特征值/特征向量，将成为经典的 syevd/heevd 例程；range = MCSOLVER_EIG_RANGE_V：将找到半开区间 (vl,vu] 中的所有特征值/特征向量；range = MCSOLVER_EIG_RANGE_I：将找到第 il 到第 iu 个特征值/特征向量；

下页继续

表 2.28 – 续上页

参数	内存	in/out	含义
uplo	host	input	指定存储 A 和 B 的哪个部分。uplo = MCBLAS_FILL_MODE_LOWER：存储 A 和 B 的下三角部分。uplo = MCBLAS_FILL_MODE_UPPER：存储 A 和 B 的上三角部分。
n	host	input	矩阵 A 和 B 的行数（或列数）。
A	device	in/out	维度为 lda * n 的 <type> 数组，其中 lda 不小于 max(1, n)。如果 uplo = MCBLAS_FILL_MODE_UPPER，则 A 的前 n * n 个上三角部分包含矩阵 A 的上三角部分。如果 uplo = MCBLAS_FILL_MODE_LOWER，则 A 的前 n * n 个下三角部分包含矩阵 A 的下三角部分。在退出时，如果 jobzv = vMCSOLVER_EIG_MODE_VECTOR，且 devInfo = 0，则 A 包含矩阵 A 的正交特征向量。如果 jobz = MCSOLVER_EIG_MODE_NOVECTOR，则 A 的内容将被清空。
lda	host	input	用于存储矩阵 A 的二维数组的前导维度。lda 不小于 max(1,n)。
B	device	in/out	维度为 ldb * n 的 <type> 数组。如果 uplo = MCBLAS_FILL_MODE_UPPER，则 B 的前 n * n 个上三角部分包含矩阵 B 的上三角部分。如果 uplo = MCBLAS_FILL_MODE_LOWER，则 B 的前 n * n 个下三角部分包含矩阵 B 的下三角部分。在退出时，如果 devInfo 小于 n，则 B 将被 Cholesky 分解的三角因子 U 或 L 覆盖。
ldb	host	input	用于存储矩阵 B 的二维数组的前导维度。ldb 不小于 max(1,n)。
vl,vu	host	input	实数值浮点或双精度浮点，分别用于 (C, S) 或 (Z, D) 精度。如果 range = MCSOLVER_EIG_RANGE_V, 将搜索特征值的区间的下界和上界。vl > vu。如果 range = MCSOLVER_EIG_RANGE_ALL 或 range = MCSOLVER_EIG_RANGE_I，则不会被引用。请注意，如果特征值非常接近，众所周知，两个不同的特征值计算程序可能会在同一区间内找到稍微不同数量的特征值。这是因为不同的特征值算法，甚至是相同的算法但不同的运行，可能会在接近机器精度的舍入误差范围内找到特征值。因此，如果用户想要确保不会在区间边界内错过任何特征值，我们建议用户将 epsilon(机器精度) 减去/加上区间边界，例如（vl = vl - eps ,vu = vu + eps）。对于来自 mcSOLVER 或 LAPACK 的任何选择性例程，此建议都是有效的。
il,iu	host	input	整数。如果 range = MCSOLVER_EIG_RANGE_I，则返回最小和最大特征值的索引（按升序排列）。如果 n > 0,1<= il <= iu <= n；如果 n = 0，则 il = 1 且 iu = 0。如果 range = MCSOLVER_EIG_RANGE_ALL 或 range = MCSOLVER_EIG_RANGE_V，则不会被引用。
h_meig	host	output	整数。找到的特征值的总数。0 <= h_meig <= n。如果 range = MCSOLVER_EIG_RANGE_ALL，则 h_meig = n；如果 range = MCSOLVER_EIG_RANGE_I，则 h_meig = iu-il+1。
W	device	output	维度为 n 的实数数组。矩阵 A 的特征值，以使得 W(i) >= W(i+1)。
work	device	in/out	工作空间，大小为 lwork 的 <type> 数组。
lwork	host	input	由 sygvdx_bufferSize 返回的 work 的大小。
devInfo	device	output	如果 devInfo = 0，操作成功。如果 devInfo = -i，则第 i 个参数错误（不包括句柄）。如果 devInfo = i(>0)，则 devInfov 表示 potrf 或 syevd 是错误的。

状态返回

MCSOLVER_STATUS_SUCCESS	操作成功完成。
MCSOLVER_STATUS_NOT_INITIALIZED	库未初始化。
MCSOLVER_STATUS_INVALID_VALUE	传递了无效的参数（n<0，或 lda<max(1,n)，或 ldb<max(1,n)，或 itype 不是 MCSOLVER_EIG_TYPE_1 或 MCSOLVER_EIG_TYPE_2 或 MCSOLVER_EIG_TYPE_3 或 jobz 不是 MCSOLVER_EIG_MODE_NOVECTOR 或 MCSOLVER_EIG_MODE_VECTORL，或 range 不是 MCSOLVER_EIG_RANGE_ALL 或 MCSOLVER_EIG_RANGE_V 或 MCSOLVER_EIG_RANGE_I，或 uplo 不是 MCBLAS_FILL_MODE_LOWER 或 MCBLAS_FILL_MODE_UPPER）。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.3.17 mcsolverDn<t>syevj()

以下辅助函数可以计算所需的预分配缓冲区大小：

```
mcsolverStatus_t
 mcsolverDnSsyevj_bufferSize(
     mcsolverDnHandle_t handle,
     mcsolverEigMode_t jobz,
     mcblasFillMode_t uplo,
     int n,
     const float *A,
     int lda,
     const float *W,
     int *lwork,
     syevjInfo_t params);

mcsolverStatus_t
 mcsolverDnDsyevj_bufferSize(
     mcsolverDnHandle_t handle,
     mcsolverEigMode_t jobz,
     mcblasFillMode_t uplo,
     int n,
     const double *A,
     int lda,
     const double *W,
     int *lwork,
     syevjInfo_t params);

mcsolverStatus_t
 mcsolverDnCheevj_bufferSize(
     mcsolverDnHandle_t handle,
     mcsolverEigMode_t jobz,
     mcblasFillMode_t uplo,
     int n,
     const mcComplex *A,
     int lda,
     const float *W,
     int *lwork,
     syevjInfo_t params);
```

(下页继续)

(续上页)

```
mcsolverStatus_t
 mcsolverDnZheevj_bufferSize(
     mcsolverDnHandle_t handle,
     mcsolverEigMode_t jobz,
     mcblasFillMode_t uplo,
     int n,
     const mcDoubleComplex *A,
     int lda,
     const double *W,
     int *lwork,
     syevjInfo_t params);
```

S 和 D 数据类型分别为单精度和双精度实数。

```
mcsolverStatus_t
 mcsolverDnSsyevj(
     mcsolverDnHandle_t handle,
     mcsolverEigMode_t jobz,
     mcblasFillMode_t uplo,
     int n,
     float *A,
     int lda,
     float *W,
     float *work,
     int lwork,
     int *info,
     syevjInfo_t params);

mcsolverStatus_t
 mcsolverDnDsyevj(
     mcsolverDnHandle_t handle,
     mcsolverEigMode_t jobz,
     mcblasFillMode_t uplo,
     int n,
     double *A,
     int lda,
     double *W,
     double *work,
     int lwork,
     int *info,
     syevjInfo_t params);
```

C 和 Z 数据类型分别为单精度和双精度的复数。

```
mcsolverStatus_t
 mcsolverDnCheevj(
     mcsolverDnHandle_t handle,
     mcsolverEigMode_t jobz,
     mcblasFillMode_t uplo,
     int n,
     mcComplex *A,
     int lda,
     float *W,
     mcComplex *work,
     int lwork,
     int *info,
```

(下页继续)

(续上页)

```
     syevjInfo_t params);

mcsolverStatus_t
 mcsolverDnZheevj(
     mcsolverDnHandle_t handle,
     mcsolverEigMode_t jobz,
     mcblasFillMode_t uplo,
     int n,
     mcDoubleComplex *A,
     int lda,
     double *W,
     mcDoubleComplex *work,
     int lwork,
     int *info,
     syevjInfo_t params);
```

这个函数计算对称（厄米特）n×n 矩阵 A 的特征值和特征向量。标准的对称特征值问题是：

$$A * Q = Q * \Lambda$$

其中 Λ 是一个实的 n×n 对角矩阵,Q 是一个 n×n 的酉矩阵，Λ 的对角元素按升序排列，它们是矩阵 A 的特征值。

syevj 与 syevd 具有相同的功能。不同之处在于，syevd 使用 QR 算法，而 syevj 使用 Jacobi 方法。Jacobi 方法的并行性使得 GPU 在中小型矩阵上具有更好的性能。此外，用户可以配置 syevj 以达到一定的精度近似。

它的工作原理是什么呢?

syevj 通过迭代生成一系列酉矩阵，将矩阵 A 转化为以下形式：

$$V^H * A * V = W + E$$

其中 W 是对角矩阵，E 是没有对角元的对称矩阵。

在迭代过程中，E 的 Frobenius 范数单调递减。当 E 趋近于零时，W 就是特征值的集合。实际上,Jacobi 方法会在以下情况停止：

其中 eps 是给定的容限值。

syevj 有两个参数用于控制精度。第一个参数是容限值 (eps)。默认值是机器精度，但用户可以使用函数 mcsolverDnXsyevjSetTolerance 来设置预先的容限值。第二个参数是最大扫描次数, 它控制 Jacobi 方法的迭代次数。默认值是 100 ，但用户可以使用函数 mcsolverDnXsyevjSetMaxSweeps 来设置适当的上限。实验表明, 进行 15 次扫描足以收敛到机器精度。syevj 会在满足容限值或达到最大扫描次数时停止。

Jacobi 方法具有二次收敛性, 因此精度与扫描次数不成比例。为了保证一定的精度，用户应该只配置容限值。

在完成 syevj 后，用户可以通过函数 mcsolverDnXsyevjGetResidual 查询残差，并通过函数 mcsolverDnXsyevjGetSweeps 查询执行的扫描次数。然而，用户需要知道残差是 E 的 Frobenius 范数，而不是单个特征值的精度。

与 syevd 一样，用户必须提供由输入参数 work 指向的工作空间。输入参数 lwork 表示工作空间的大小，可以通过 syevj_bufferSize() 函数返回。

如果输出参数 info = -i （小于零），则第 i 个参数错误（不包括句柄）。如果 info = n+1 ，则 syevj 在给定的容限值和最大扫描次数下无法收敛。

如果用户设置了不合适的容限值，syevj 可能无法收敛。例如，容限值不应小于机器精度。

如果 jobz = MCSOLVER_EIG_MODE_VECTOR，A 包含正交特征向量 V 。

API of syevj

参数	内存	in/out	含义
handle	host	input	mcsolverDN 库上下文的句柄。
jobz	host	input	指定选项以仅计算特征值或计算特征值和特征向量：jobz = MCSOLVER_EIG_MODE_NOVECTOR：仅计算特征值；jobz = MCSOLVER_EIG_MODE_VECTOR：计算特征值和特征向量。
uplo	host	input	指定存储矩阵 A 的哪个部分。uplo = MCBLAS_FILL_MODE_LOWER：存储矩阵 A 的下三角部分；uplo = MCBLAS_FILL_MODE_UPPER：存储矩阵 A 的上三角部分。
n	host	input	矩阵 A 的行数（或列数）。
A	device	in/out	维度为 lda * n 的 <type> 数组，其中 lda 不小于 max(1, n)。如果 uplo = MCBLAS_FILL_MODE_UPPER，则矩阵 A 的前 n 行、前 n 列的上三角部分包含了矩阵 A 的上三角部分。如果 uplo = MCBLAS_FILL_MODE_LOWER，则矩阵 A 的前 n 行、前 n 列的下三角部分包含了矩阵 A 的下三角部分。在退出时，如果 jobz = MCSOLVER_EIG_MODE_VECTOR，且 info = 0，则矩阵 A 包含矩阵 A 的正交特征向量。如果 jobz = MCSOLVER_EIG_MODE_NOVECTOR，则矩阵 A 的内容会被清空。
lda	host	input	用于存储矩阵 A 的二维数组的前导维度。
W	device	output	维度为 n 的实数组。矩阵 A 的特征值，按升序排列，即排序为使 W(i) <= W(i+1)。
work	device	in/out	工作空间，大小为 lwork 的 <type> 数组。
lwork	host	input	由 syevj_bufferSize 返回的 work 的大小。
info	device	output	如果 info = 0，操作成功。如果 info = -i，则第 i 个参数有误（不包括句柄）。如果 info = n+1，则 syevj 在给定的容差和最大迭代次数下不收敛。
params	host	in/out	结构体，填充有 Jacobi 算法的参数和 syevj 的结果。

状态返回

MCSOLVER_STATUS_SUCCESS	操作成功完成
MCSOLVER_STATUS_NOT_INITIALIZED	未初始化库。
MCSOLVER_STATUS_INVALID_VALUE	传递了无效的参数（n<0，或 lda<max(1,n)，或 jobz 不是 MCSOLVER_EIG_MODE_NOVECTOR 或 MCSOLVER_EIG_MODE_VECTOR，或 uplo 不是 MCBLAS_FILL_MODE_LOWER 或 MCBLAS_FILL_MODE_UPPER）。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.3.18 mcsolverDn<t>sygvj()

下面的辅助函数可以计算预分配缓冲区所需的大小。

```
mcsolverStatus_t
 mcsolverDnSsygvj_bufferSize(
     mcsolverDnHandle_t handle,
     mcsolverEigType_t itype,
     mcsolverEigMode_t jobz,
     mcblasFillMode_t uplo,
     int n,
     const float *A,
     int lda,
     const float *B,
     int ldb,
     const float *W,
```

(下页继续)

(续上页)

```
    int *lwork,
    syevjInfo_t params);

mcsolverStatus_t
 mcsolverDnDsygvj_bufferSize(
    mcsolverDnHandle_t handle,
    mcsolverEigType_t itype,
    mcsolverEigMode_t jobz,
    mcblasFillMode_t uplo,
    int n,
    const double *A,
    int lda,
    const double *B,
    int ldb,
    const double *W,
    int *lwork,
    syevjInfo_t params);

mcsolverStatus_t
 mcsolverDnChegvj_bufferSize(
    mcsolverDnHandle_t handle,
    mcsolverEigType_t itype,
    mcsolverEigMode_t jobz,
    mcblasFillMode_t uplo,
    int n,
    const mcComplex *A,
    int lda,
    const mcComplex *B,
    int ldb,
    const float *W,
    int *lwork,
    syevjInfo_t params);

mcsolverStatus_t
 mcsolverDnZhegvj_bufferSize(
    mcsolverDnHandle_t handle,
    mcsolverEigType_t itype,
    mcsolverEigMode_t jobz,
    mcblasFillMode_t uplo,
    int n,
    const mcDoubleComplex *A,
    int lda,
    const mcDoubleComplex *B,
    int ldb,
    const double *W,
    int *lwork,
    syevjInfo_t params);
```

S 数据类型和 D 数据类型分别表示单精度和双精度实数。

```
mcsolverStatus_t
 mcsolverDnSsygvj(
    mcsolverDnHandle_t handle,
    mcsolverEigType_t itype,
    mcsolverEigMode_t jobz,
    mcblasFillMode_t uplo,
```

(下页继续)

(续上页)

```
    int n,
    float *A,
    int lda,
    float *B,
    int ldb,
    float *W,
    float *work,
    int lwork,
    int *info,
    syevjInfo_t params);

mcsolverStatus_t
 mcsolverDnDsygvj(
    mcsolverDnHandle_t handle,
    mcsolverEigType_t itype,
    mcsolverEigMode_t jobz,
    mcblasFillMode_t uplo,
    int n,
    double *A,
    int lda,
    double *B,
    int ldb,
    double *W,
    double *work,
    int lwork,
    int *info,
    syevjInfo_t params);
```

C 数据类型和 Z 数据类型分别表示单精度和双精度复数。

```
mcsolverStatus_t
 mcsolverDnChegvj(
    mcsolverDnHandle_t handle,
    mcsolverEigType_t itype,
    mcsolverEigMode_t jobz,
    mcblasFillMode_t uplo,
    int n,
    mcComplex *A,
    int lda,
    mcComplex *B,
    int ldb,
    float *W,
    mcComplex *work,
    int lwork,
    int *info,
    syevjInfo_t params);

mcsolverStatus_t
 mcsolverDnZhegvj(
    mcsolverDnHandle_t handle,
    mcsolverEigType_t itype,
    mcsolverEigMode_t jobz,
    mcblasFillMode_t uplo,
    int n,
    mcDoubleComplex *A,
    int lda,
```

(下页继续)

(续上页)

```
    mcDoubleComplex *B,
    int ldb,
    double *W,
    mcDoubleComplex *work,
    int lwork,
    int *info,
    syevjInfo_t params);
```

该函数计算对称（厄米特）的 n×n 矩阵对（A、B）的特征值和特征向量。广义对称正定特征值问题是指：

该函数具有与 sygvd 相同的功能，唯一的区别在于 sygvd 中的 syevd 被 sygvj 中的 syevj 所替代。因此，sygvj 继承了 syevj 的属性，用户可以使用 mcsolverDnXsyevjSetTolerance 和 mcsolverDnXsyevjSetMaxSweeps 来配置容差和最大迭代次数。

然而，与 syevj 不同，sygvj 中残差的含义不同。sygvj 首先计算矩阵 B 的 Cholesky 分解，

$$B = L * L^H$$

将问题转化为标准特征值问题，然后调用 syevj 函数。

例如，类型 I 的标准特征值问题为

$$M * Q = Q * \Lambda$$

其中矩阵 M 是对称的。

$$M = L^{-1} * A * L^{-H}$$

残差是在矩阵 M 上使用 syevj 的结果，而不是 A。

用户需要提供由输入参数 work 指向的工作空间。输入参数 lwork 表示工作空间的大小，并且由 sygvj_bufferSize() 返回。

如果输出参数 info = -i （小于零），表示第 i 个参数错误（不包括句柄）。如果 info = i （i > 0 且 i <= n），则 B 不是正定的，无法完成 B 的分解，并且未计算任何特征值或特征向量。如果 info = n+1，则 syevj 在给定的容差和最大迭代次数下不收敛。在这种情况下，仍会计算特征值和特征向量，因为不收敛是由于最大迭代次数的容差不当所造成的。

如果 jobz = MCSOLVER_EIG_MODE_VECTOR，那么矩阵 A 包含正交特征向量 V。

API of sygvj

参数	内存	in/out	含义
handle	host	input	mcsolverDN 库上下文的句柄。
itype	host	input	指定要解决的问题类型：itype = MCSOLVER_EIG_TYPE_1: A*x = (lambda)*B*x。itype = MCSOLVER_EIG_TYPE_2: A*B*x = (lambda)*x. itype = MCSOLVER_EIG_TYPE_3: B*A*x = (lambda)*x。
jobz	host	input	指定选项以仅计算特征值或计算特征值对：jobz = MCSOLVER_EIG_MODE_NOVECTOR：仅计算特征值；jobz = MCSOLVER_EIG_MODE_VECTOR：计算特征值和特征向量。
uplo	host	input	指定 A 和 B 的哪个部分被存储。uplo = MCBLAS_FILL_MODE_LOWER：存储 A 和 B 的下三角部分。uplo = MCBLAS_FILL_MODE_UPPER：存储 A 和 B 的上三角部分。
n	host	input	矩阵 Aj 的行数（或列数）。
A	device	in/out	<type> lda * n 维数组，其中 lda 不小于 max(1,n)。如果 uplo = MCBLAS_FILL_MODE_UPPER，矩阵 A 的 n * n 的上三角部分包含矩阵 A 的上三角部分。如果 uplo = MCBLAS_FILL_MODE_LOWER，矩阵 A 的 n * n 的下三角部分包含矩阵 A 的下三角部分。退出时，如果 jobz = MCSOLVER_EIG_MODE_VECTOR，且 info = 0，A 包含矩阵 A 的正交特征向量。如果 jobz = MCSOLVER_EIG_MODE_NOVECTOR，则 A 的内容被销毁。

下页继续

表 2.30 – 续上页

参数	内存	in/out	含义
lda	host	input	用于存储矩阵 A 的二维数组的前导维度。lda 不小于 max(1,n)。
B	device	in/out	<type> 维度为 ldb * n 的数组。如果 uplo = MCBLAS_FILL_MODE_UPPER，则矩阵 B 的 n * n 的上三角部分包含在 B 的上三角部分。如果 uplo = MCBLAS_FILL_MODE_LOWER，则矩阵 B 的 n * n 的下三角部分包含在 B 的下三角部分。完成后，如果 info 小于 n，则 B 将被从 B 的 Cholesky 分解中的三角因子 U 或 L 覆盖。
ldb	host	input	用于存储矩阵 B 的二维数组的前导维度。ldb 不小于 max(1,n)。
W	device	output	维度为 n 的实数数组。数组 W 包含了矩阵 A 的特征值，排序使得 W(i) >= W(i+1)。
work	device	in/out	工作空间，大小为 lwork 的 <type> 数组。
lwork	host	input	work 的大小，由 sygvj_bufferSize 返回。
info	device	output	如果 info = 0，表示操作成功。如果 info = -i，表示第 i 个参数错误（不包括句柄）。如果 info = i(>0)，则说明要么 B 不是正定的，要么 syevj （由 sygvj 调用）不收敛。

返回状态

MCSOLVER_STATUS_SUCCESS	操作顺利完成。
MCSOLVER_STATUS_NOT_INITIALIZED	库未初始化。
MCSOLVER_STATUS_INVALID_VALUE	传递了无效的参数（n<0，或 lda<max(1,n)，或 ldb<max(1,n)，或 itype 不是 1、2 或 3，或 jobz 不是 MCSOLVER_EIG_MODE_NOVECTOR 或 MCSOLVER_EIG_MODE_VECTOR，或 uplo 不是 MCBLAS_FILL_MODE_LOWER 或 MCBLAS_FILL_MODE_UPPER）。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.3.19 mcsolverDn<t>syevjBatched()

下面的辅助函数可以计算预分配缓冲区所需的大小。

```
mcsolverStatus_t
 mcsolverDnSsyevjBatched_bufferSize(
     mcsolverDnHandle_t handle,
     mcsolverEigMode_t jobz,
     mcblasFillMode_t uplo,
     int n,
     const float *A,
     int lda,
     const float *W,
     int *lwork,
     syevjInfo_t params,
     int batchSize
     );

mcsolverStatus_t
 mcsolverDnDsyevjBatched_bufferSize(
     mcsolverDnHandle_t handle,
     mcsolverEigMode_t jobz,
     mcblasFillMode_t uplo,
     int n,
```

(下页继续)

(续上页)

```
    const double *A,
    int lda,
    const double *W,
    int *lwork,
    syevjInfo_t params,
    int batchSize
    );

mcsolverStatus_t
 mcsolverDnCheevjBatched_bufferSize(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    mcblasFillMode_t uplo,
    int n,
    const mcComplex *A,
    int lda,
    const float *W,
    int *lwork,
    syevjInfo_t params,
    int batchSize
    );

mcsolverStatus_t
 mcsolverDnZheevjBatched_bufferSize(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    mcblasFillMode_t uplo,
    int n,
    const mcDoubleComplex *A,
    int lda,
    const double *W,
    int *lwork,
    syevjInfo_t params,
    int batchSize
    );
```

S 数据类型和 D 数据类型分别表示单精度和双精度实数。

```
mcsolverStatus_t
 mcsolverDnSsyevjBatched(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    mcblasFillMode_t uplo,
    int n,
    float *A,
    int lda,
    float *W,
    float *work,
    int lwork,
    int *info,
    syevjInfo_t params,
    int batchSize
    );

mcsolverStatus_t
 mcsolverDnDsyevjBatched(
```

(下页继续)

(续上页)

```
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    mcblasFillMode_t uplo,
    int n,
    double *A,
    int lda,
    double *W,
    double *work,
    int lwork,
    int *info,
    syevjInfo_t params,
    int batchSize
    );
```

C 数据类型和 Z 数据类型分别表示单精度和双精度复数。

```
mcsolverStatus_t
 mcsolverDnCheevjBatched(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    mcblasFillMode_t uplo,
    int n,
    mcComplex *A,
    int lda,
    float *W,
    mcComplex *work,
    int lwork,
    int *info,
    syevjInfo_t params,
    int batchSize
    );

mcsolverStatus_t
 mcsolverDnZheevjBatched(
    mcsolverDnHandle_t handle,
    mcsolverEigMode_t jobz,
    mcblasFillMode_t uplo,
    int n,
    mcDoubleComplex *A,
    int lda,
    double *W,
    mcDoubleComplex *work,
    int lwork,
    int *info,
    syevjInfo_t params,
    int batchSize
    );
```

该函数计算一系列对称（厄米特）的 n×n 矩阵的特征值和特征向量。

$$A_j * Q_j = Q_j * \Lambda_j$$

其中 A 是一个 n×n 对角实矩阵，Q 是一个 n×n 酉矩阵。

`syevjBatched` 对每个矩阵执行 `syevj` 操作。它要求所有矩阵具有相同的大小 `n`，并以连续的方式打包。

每个矩阵都是按列主序存储，其前导维度为 `lda`，因此随机访问的公式是。

参数 W 也以连续方式包含每个矩阵的特征值。

除了公差和最大迭代次数之外，syevjBatched 函数可以按升序（默认）或按原顺序（不排序）对特征值进行排序，通过函数 mcsolverDnXsyevjSetSortEig 进行设置。如果用户将多个小矩阵打包成一个大矩阵的对角块，不进行排序的选项可以分离这些小矩阵的频谱。

syevjBatched 函数无法通过 mcsolverDnXsyevjGetResidual 和 mcsolverDnXsyevjGetSweeps 函数报告残差和执行的迭代次数。对以上两个函数的任何调用都将返回 MCSOLVER_STATUS_NOT_SUPPORTED。用户需要显式计算残差。

用户需要提供由输入参数 work 指向的工作空间。输入参数 lwork 表示工作空间的大小，并且可以通过 syevjBatched_bufferSize() 返回。

输出参数 info 是一个大小为 batchSize 的整数数组。如果函数返回 MCSOLVER_STATUS_INVALID_VALUE，则第一个元素 info[0]=-i （小于零）表示第 i 个参数错误（不包括句柄）。另外，如果 info[i] = n+1，表示 syevjBatched 在给定的公差和最大迭代次数下未收敛于第 i 个矩阵。

如果 jobz = MCSOLVER_EIG_MODE_VECTOR，则包含正交归一化的特征向量。

API of syevjBatched

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库上下文的句柄。
jobz	host	input	根据选项来指定是仅计算特征值还是同时计算特征值和特征向量：jobz = MCSOLVER_EIG_MODE_NOVECTOR：仅计算特征值；jobz = MCSOLVER_EIG_MODE_VECTOR：计算特征值和特征向量。
uplo	host	input	指定存储 Aj 的哪个部分。uplo = MCBLAS_FILL_MODE_LOWER：存储 Aj 的下三角部分。uplo = MCBLAS_FILL_MODE_UPPER：存储 Aj 的上三角部分。
n	host	input	矩阵 Aj 的行数（或列数）。
A	device	in/out	<type> 具有 lda * n * batchSize 维度的数组，其中 lda 不小于 max(1,n)。如果 uplo = MCBLAS_FILL_MODE_UPPER，则 Aj 的前 n * n 的上三角部分包含矩阵 Aj 的上三角部分。如果 uplo = MCBLAS_FILL_MODE_LOWER，则 Aj 的前 n * n 的下三角部分包含矩阵 Aj 的下三角部分。执行完毕后，如果 jobz = MCSOLVER_EIG_MODE_VECTOR，并且 info[j] = 0，Aj 中包含矩阵 Aj 的正交特征向量。如果 jobz = MCSOLVER_EIG_MODE_NOVECTOR，则 Aj 的内容将被销毁。
lda	host	input	前导维度，用于存储矩阵 Aj 的二维数组。
W	device	output	一个维度为 n*batchSize 的实数数组。它按升序或非排序顺序存储了矩阵 Aj 的特征值。
work	device	in/out	<type> 工作空间，大小为 lwork 的数组。
lwork	host	input	work 的大小，由 syevjBatched_bufferSize 返回。
info	device	output	一个维度为 batchSize 的整数数组。如果返回 MCSOLVER_STATUS_INVALID_VALUE，则 info[0] = -i （小于零）表示第 i 个参数错误（不包括句柄）。否则，如果 info[i] = 0，则表示操作成功。如果 info[i] = n+1，则表示在给定的容差和最大迭代次数下，syevjBatched 未收敛于第 i 个矩阵。
params	host	in/out	填充有 Jacobi 算法参数的结构体。
batchSize	host	input	矩阵的数量。

返回状态

MCSOLVER_STATUS_SUCCESS	操作顺利完成。
MCSOLVER_STATUS_NOT_INITIALIZED	库未初始化。
MCSOLVER_STATUS_INVALID_VALUE	传递的参数无效（n<0，或 lda<max(1,n)，或 jobz 不是 MCSOLVER_EIG_MODE_NOVECTOR 或 MCSOLVER_EIG_MODE_VECTOR，或 uplo 不是 MCBLAS_FILL_MODE_LOWER 或 MCBLAS_FILL_MODE_UPPER），或 batchSize<0。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.4 稠密线性求解器参考 (64-bit API)

本节介绍 mcsolverDN 的 64 位线性求解器 API，包括 Cholesky 分解、使用部分主元消元法的 LU 分解和 QR 分解。

2.4.4.1 mcsolverDnXpotrf()

下面的辅助函数可以计算所需的预分配缓冲区大小。

```
mcsolverStatus_t
mcsolverDnXpotrf_bufferSize(
    mcsolverDnHandle_t handle,
    mcsolverDnParams_t params,
    mcblasFillMode_t uplo,
    int64_t n,
    macaDataType dataTypeA,
    const void *A,
    int64_t lda,
    macaDataType computeType,
    size_t *workspaceInBytesOnDevice,
    size_t *workspaceInBytesOnHost)
```

例程如下：

```
mcsolverStatus_t
mcsolverDnXpotrf(
    mcsolverDnHandle_t handle,
    mcsolverDnParams_t params,
    mcblasFillMode_t uplo,
    int64_t n,
    macaDataType dataTypeA,
    void *A,
    int64_t lda,
    macaDataType computeType,
    void *bufferOnDevice,
    size_t workspaceInBytesOnDevice,
    void *bufferOnHost,
    size_t workspaceInBytesOnHost,
    int *info )
```

使用通用 API 接口计算厄米特正定矩阵的 Cholesky 分解。

A 是一个 n×n 厄米特矩阵，只有下半部分或上半部分才有意义。输入参数 uplo 指示函数使用矩阵的哪一部分，其他部分保持不变。

如果输入参数 uplo 为 MCBLAS_FILL_MODE_LOWER，仅处理 A 的下三角部分，并由下三角 Cholesky 因子 L 代替。

$$A = L * L^H$$

如果输入参数 uplo 为 MCBLAS_FILL_MODE_UPPER，仅处理 A 的上三角部分，并由上三角 Cholesky 因子 U 代替。

$$A = U^H * U$$

用户必须提供由输入参数 bufferOnDevice 和 bufferOnHost 指向的设备和主机工作空间。输入参数 workspaceInBytesOnDevice (以及 workspaceInBytesOnHost) 是设备 (和主机) 工作空间的字节大小，并且由 mcsolverDnXpotrf_bufferSize() 返回。

如果 Cholesky 分解失败，即 A 的导数是不正定的，或者等价于 L 或 U 的对角元素不是实数，输出参数 info 将指出 A 的导数不是正定的。

如果输出参数 info=-i (小于 0)，则第 i 个参数是错误的 (不包含句柄)。

目前，mcsolverDnXpotrf 仅支持默认的算法。

mcsolverDnXpotrf 支持的算法表

mcsolver_ALG_0 or NULL	默认算法

API of potrf

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库上下文的句柄。
params	host	input	由 mcsolverDnSetAdvOptions 收集的信息的结构体。
uplo	host	input	指示矩阵 A 的下半部分或上半部分是否被存储，另一部分未被引用。
n	host	input	矩阵 A 的行数和列数。
dataTypeA	host	in	数组 A 的数据类型。
A	device	in/out	具有 lda * n 维的数组，其中 lda 不小于 max(1,n) 。
lda	host	input	用于存储矩阵 A 的二维数组的前导维度大小。
computeType	host	in	计算的数据类型。
bufferOnDevice	device	in/out	设备工作空间。类型为 void 、大小为 workspaceInBytesOnDevice 字节的数组。
workspaceInBytesOnDevice	host	input	由 mcsolverDnXpotrf_bufferSize 返回的 bufferOnDevice 的字节大小。
bufferOnHost	host	in/out	主机工作空间。类型为 void 、大小为 workspaceInBytesOnHost 字节的数组。
workspaceInBytesOnHost	host	input	由 mcsolverDnXpotrf_bufferSize 返回的 bufferOnHost 的字节大小。
info	device	output	如果 info = 0 ，则 Cholesky 分解成功。如果 info = -i ，表示第 i 个参数错误（不包括句柄）。如果 info = i ，则大小为 i 的前导子阵不是正定的。

通用 API 有两种不同的类型，dataTypeA 是矩阵 A 的数据类型，computeType 是操作的计算类型。mcsolverDnXpotrf 仅支持以下四种组合。

数据类型和计算类型的有效组合

DataTypeA	ComputeType	Meaning
MACA_R_32F	MACA_R_32F	SPOTRF
MACA_R_64F	MACA_R_64F	DPOTRF
MACA_C_32F	MACA_C_32F	CPOTRF
MACA_C_64F	MACA_C_64F	ZPOTRF

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。
MCSOLVER_STATUS_NOT_INITIALIZED	库未进行初始化。
MCSOLVER_STATUS_INVALID_VALUE	传递的参数无效 (n<0 或 lda<max(1,n))。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.4.2 mcsolverDnXpotrs()

```
mcsolverStatus_t
mcsolverDnXpotrs(
    mcsolverDnHandle_t handle,
    mcsolverDnParams_t params,
    mcblasFillMode_t uplo,
    int64_t n,
    int64_t nrhs,
    macaDataType dataTypeA,
    const void *A,
    int64_t lda,
    macaDataType dataTypeB,
    void *B,
    int64_t ldb,
    int *info)
```

这个函数解决了一个线性方程组。

$$A * X = B$$

A 是一个 n×n 厄米特矩阵，在使用通用 API 接口时，只有下三角部分或上三角部分是有意义的。输入参数 uplo 指示函数使用矩阵的哪一部分，其他部分保持不变。

用户必须首先调用 mcsolverDnXpotrf 对矩阵 A 进行因式分解。

- 如果输入参数 uplo = MCBLAS_FILL_MODE_LOWER，A 是下三角 Cholesky 因子 L 对应于 $A = L * L^H$ 。
- 如果输入参数 uplo = MCBLAS_FILL_MODE_UPPER，A 是上三角 Cholesky 因子 U 对应于 $A = U^H * U$ 。

该操作是就地进行的，即矩阵 X 覆盖矩阵 B 具有相同的前导维度 ldb 。如果输出参数 Info = -i （小于零），则第 i 个参数是错误的（不包含句柄）。目前，mcsolverDnXpotrs 只支持默认算法。

mcsolverDnXpotrs 支持的算法表

mcsolver_ALG_0 or NULL	默认算法

API of potrs

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库上下文的句柄。

下页继续

表 2.33 – 续上页

参数	内存	In/out	含义
params	host	input	由 mcsolverDnSetAdvOptions 收集的信息的结构体。
uplo	host	input	指示矩阵 A 的下半部分或上半部分是否被存储，另一部分未被引用。
n	host	input	矩阵 A 的行数和列数。
nrhs	host	input	矩阵 X 和 B 的列数。
dataTypeA	host	in	数组 A 的数据类型。
A	device	input	维度为 lda * n 的数组，其中 lda 不小于 max(1, n)。数组 A 可以是下三角形的 cholesky 因子 L 或上三角形的 Cholesky 因子 U。
lda	host	input	用于存储矩阵 A 的二维数组的前导维度大小。
dataTypeB	host	in	数组 B 的数据类型。
B	device	in/ out	维度为 ldb * nrhs 的数组。其中 ldb 不小于 max(1, n)。作为输入，数组 B 是右侧矩阵。作为输出，数组 B 是解矩阵。
info	device	output	如果 info = 0，则 Cholesky 分解成功。如果 info = -i，表示第 i 个参数错误（不包括句柄）。

通用 API 有两种不同的类型。dataTypeA 是矩阵 A 的数据类型。dataTypeB 是矩阵 B 的数据类型。mcsolverDnXpotrs 仅支持以下 4 种组合。

数据类型和计算类型的有效组合

dataTypeA	dataTypeB	Meaning
MACA_R_32F	MACA_R_32F	SPOTRS
MACA_R_64F	MACA_R_64F	DPOTRS
MACA_C_32F	MACA_C_32F	CPOTRS
MACA_C_64F	MACA_C_64F	ZPOTRS

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。
MCSOLVER_STATUS_NOT_INITIALIZED	库未进行初始化。
MCSOLVER_STATUS_INVALID_VALUE	传递的参数无效 (n<0 或 lda<max(1,n))。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.4.3 mcsolverDnXgetrf()

以下辅助函数可以计算预分配缓冲区所需的大小。

```
mcsolverStatus_t mcsolverDnXgetrf_bufferSize(
    mcsolverDnHandle_t handle,
    mcsolverDnParams_t params,
    int64_t m,
    int64_t n,
    macaDataType dataTypeA,
    const void *A,
    int64_t lda,
    macaDataType computeType,
    size_t *workspaceInBytesOnDevice,
    size_t *workspaceInBytesOnHost)
```

以下函数

```
mcsolverStatus_t
mcsolverDnXgetrf(
    mcsolverDnHandle_t handle,
    mcsolverDnParams_t params,
    int64_t m,
    int64_t n,
    macaDataType dataTypeA,
    void *A,
    int64_t lda,
    int64_t *ipiv,
    macaDataType computeType,
    void *bufferOnDevice,
    size_t workspaceInBytesOnDevice,
    void *bufferOnHost,
    size_t workspaceInBytesOnHost,
    int *info )
```

计算一个 m×n 矩阵的 LU 分解。

$$P * A = L * U$$

其中 A 是一个 m×n 矩阵，P 是一个置换矩阵，L 是一个具有单位对角线的下三角矩阵，U 是一个上三角矩阵，使用通用 API 接口进行计算。

如果 LU 分解失败，即矩阵 A(U) 是奇异的，输出参数 info=i 表示 U(i,i)=0 。

如果输出参数 Info = -i （小于零），则第 i 个参数是错误的（不包含句柄）。

如果 ipiv 为空，将不进行主元交换。分解结果为 A=L*U ，这种分解方法在数值上不稳定。

无论 LU 分解是否失败，输出参数 ipiv 都包含了主元交换的顺序，第 i 行与第 ipiv(i) 行进行了交换。

用户必须提供由输入参数 bufferOnDevice 和 bufferOnHost 指向的设备和主机工作空间。输入参数 workspaceInBytesOnDevice （和 workspaceInBytesOnHost ）是设备（和主机）工作空间的字节大小，并且可以通过 mcsolverDnXgetrf_bufferSize() 返回。

用户可以结合使用 mcsolverDnXgetrf 和 mcsolverDnGetrs 来完成线性求解器。

目前，mcsolverDnXgetrf 支持两种算法。要选择传统实现，用户必须调用 mcsolverDnSetAdvOptions 。

mcsolverDnXgetrf 支持的算法

MCSOLVER_ALG_0 or NULL	默认算法。最快的算法，需要一个有 m*n 个元素的工作空间。
MCSOLVER_ALG_1	传统实现。

mcsolverDnXgetrf_bufferSize 和 mcsolverDnXgetrf 的参数列表：

API of mcsolverDnXgetrf

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库上下文的句柄。
params	host	input	由 mcsolverDnSetAdvOptions 收集的信息的结构体。
m	host	input	矩阵 A 的行数。
n	host	input	矩阵 A 的列数。
dataTypeA	host	in	数组 A 的数据类型。
A	device	in/out	<type> 维度为 lda * n 的数组，其中 lda 不小于 max(1, m) 。
lda	host	input	用于存储矩阵 A 的二维数组的前导维度大小。

下页继续

表 2.34 – 续上页

参数	内存	In/out	含义
ipiv	device	output	大小至少为 min(m, n) 的数组，包含了主元索引。
computeType	host	in	计算的数据类型。
bufferOnDevice	device	in/out	设备工作空间。类型为 void 、大小为 workspaceInBytesOnDevice 字节的数组。
workspaceInBytesOnDevice	host	input	由 mcsolverDnXpotrf_bufferSize 返回的 bufferOnDevice 的字节大小。
bufferOnHost	host	in/out	主机工作空间。类型为 void 、大小为 workspaceInBytesOnHost 字节的数组。
workspaceInBytesOnHost	host	input	由 mcsolverDnXpotrf_bufferSize 返回的 bufferOnHost 的字节大小。
info	device	output	如果 info = 0，则 Cholesky 分解成功。如果 info = -i，表示第 i 个参数错误（不包括句柄）。如果 info = i，则 U(i,i) = 0。

通用 API 有两种不同的类型，dataTypeA 是矩阵 A 的数据类型，computeType 是操作的计算类型。mcsolverDnXgetrf 仅支持以下 4 种组合。

数据类型和计算类型的有效组合

DataTypeA	ComputeType	Meaning
MACA_R_32F	MACA_R_32F	SGETRF
MACA_R_64F	MACA_R_64F	DGETRF
MACA_C_32F	MACA_C_32F	CGETRF
MACA_C_64F	MACA_C_64F	ZGETRF

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。
MCSOLVER_STATUS_NOT_INITIALIZED	库未进行初始化。
MCSOLVER_STATUS_INVALID_VALUE	传递的参数无效 (n<0 或 lda<max(1,n))。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.4.4 mcsolverDnXgetrs()

```
mcsolverStatus_t
mcsolverDnXgetrs(
    mcsolverDnHandle_t handle,
    mcsolverDnParams_t params,
    mcblasOperation_t trans,
    int64_t n,
    int64_t nrhs,
    macaDataType dataTypeA,
    const void *A,
    int64_t lda,
    const int64_t *ipiv,
    macaDataType dataTypeB,
    void *B,
    int64_t ldb,
    int *info )
```

这个函数解决了一组包含多个右侧项的线性方程组

$$op(A) * X = B$$

其中 A 是一个 n×n 矩阵，并且通过 mcsolverDnXgetrf 进行了 LU 分解，即 A 的下三角部分是 L，上三角部分（包括对角元素）是 U。B 是使用通用 API 接口的 n×nrhs 右边矩阵。

输入参数 ipiv 是 mcsolverDnXgetrf 的输出结果。它包含了用于对右端量进行置换的主元索引。如果输出参数 Info = -i （小于零），则第 i 个参数是错误的（不包含句柄）。用户可以结合 mcsolverDnXgetrf 和 mcsolverDnXgetrs 来完成线性求解器。目前，mcsolverDnXgetrs 只支持默认算法。

mcsolverDnXpotrs 支持的算法表

mcsolver_ALG_0 or NULL	默认算法

mcsolverDnXgetrss 的输入参数列表：

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库上下文的句柄。
params	host	input	由 mcsolverDnSetAdvOptions 收集的信息的结构体。
trans	host	input	操作 op(A) 表示对矩阵 A 进行非转置或（共轭）转置。
n	host	input	矩阵 A 的行数和列数。
nrhs	host	input	右侧项的数量。
dataTypeA	host	in	数组 A 的数据类型。
A	device	input	维度为 lda * n 的数组，其中 lda 不小于 max(1, n)。数组 A 可以是下三角形的 cholesky 因子 L 或上三角形的 Cholesky 因子 U。
lda	host	input	用于存储矩阵 A 的二维数组的前导维度大小。
ipiv	device	input	大小至少为 n 的数组，包含主元索引。
dataTypeB	host	in	数组 B 的数据类型。
B	device	output	<type> 维度为 ldb * nrhs 的数组，其中 ldb 不小于 max(1,n)。
ldb	host	input	用于存储矩阵 B 的二维数组的前导维度大小。
info	device	output	如果 info = 0，操作成功。如果 info = -i，则第 i 个参数有误（不包括句柄）。

通用 API 有两种不同的类型，dataTypeA 是矩阵 A 的数据类型，computeType 是操作的计算类型。mcsolverDnXgetrs 仅支持以下四种组合。

数据类型和计算类型的有效组合

DataTypeA	ComputeType	Meaning
MACA_R_32F	MACA_R_32F	SGETRF
MACA_R_64F	MACA_R_64F	DGETRF
MACA_C_32F	MACA_C_32F	CGETRF
MACA_C_64F	MACA_C_64F	ZGETRF

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。
MCSOLVER_STATUS_NOT_INITIALIZED	库未初始化。
MCSOLVER_STATUS_INVALID_VALUE	传递的参数无效 (n<0 或 lda<max(1,n))。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.4.5 mcsolverDnXgeqrf()

下面的辅助函数可以计算所需的预分配缓冲区大小。

```
mcsolverStatus_t
mcsolverDnXgeqrf_bufferSize(
    mcsolverDnHandle_t handle,
    mcsolverDnParams_t params,
    int64_t m,
    int64_t n,
    macaDataType dataTypeA,
    const void *A,
    int64_t lda,
    macaDataType dataTypeTau,
    const void *tau,
    macaDataType computeType,
    size_t *workspaceInBytesOnDevice,
    size_t *workspaceInBytesOnHost)
```

例程如下：

```
mcsolverStatus_t mcsolverDnXgeqrf(
    mcsolverDnHandle_t handle,
    mcsolverDnParams_t params,
    int64_t m,
    int64_t n,
    macaDataType dataTypeA,
    void *A,
    int64_t lda,
    macaDataType dataTypeTau,
    void *tau,
    macaDataType computeType,
    void *bufferOnDevice,
    size_t workspaceInBytesOnDevice,
    void *bufferOnHost,
    size_t workspaceInBytesOnHost,
    int *info )
```

计算一个 m×n 矩阵的 QR 分解。

$$A = Q * R$$

其中 A 是一个 m×n 矩阵，Q 是一个 m×n 矩阵，R 是一个使用通用 API 接口的 n×n 上三角矩阵。

用户必须提供由输入参数 bufferOnDevice and bufferOnHost 指向的设备和主机工作空间。输入参数 workspaceInBytesOnDevice （和 workspaceInBytesOnHost ）是设备（和主机）工作空间的字节大小，它由 mcsolverDnXgeqrf_bufferSize() 返回。

矩阵 R 被覆盖到 A 的上三角部分，包括对角元素。

矩阵 Q 不是明确形成的，而是一系列的 Householder 向量存储在 A 的下三角部分。假设 Householder 向量的主要非零元素为 1，因此输出参数 TAU 包含了缩放因子 τ 。如果 v 是原始的 Householder 向量，q 是与 τ 对应的新的 Householder 向量，满足以下关系式:

$$I - 2 * v * v^H = I - \tau * q * q^H$$

如果输出参数 Info = -i (小于零)，则第 i 个参数是错误的 (不包含句柄)。目前，mcsolverDnXgeqrf 只支持默认算法。

mcsolverDnXgeqrf_bufferSize 和 mcsolverDnXgeqrf 的输入参数列表：

API of geqrf

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库上下文的句柄。
params	host	input	由 mcsolverDnSetAdvOptions 收集的信息的结构体。
m	host	input	矩阵 A 的行数。
n	host	input	矩阵 A 的列数。
dataTypeA	host	in	数组 A 的数据类型。
A	device	in/out	维度为 lda * n 的数组，其中 lda 不小于 max(1, m)。
lda	host	input	用于存储矩阵 A 的二维数组的前导维度大小。
TAU	device	output	维度至少为 min(m,n) 的数组。
computeType	host	in	计算的数据类型。
bufferOnDevice	device	in/out	设备工作空间。类型为 void 、大小为 workspaceInBytesOnDevice 字节的数组。
workspaceInBytesOnDevice	host	input	由 mcsolverDnXpotrf_bufferSize 返回的 bufferOnDevice 的字节大小。
bufferOnHost	host	in/out	主机工作空间。类型为 void 、大小为 workspaceInBytesOnHost 字节的数组。
workspaceInBytesOnHost	host	input	由 mcsolverDnXpotrf_bufferSize 返回的 bufferOnHost 的字节大小。
info	device	output	如果 info = 0，则 Cholesky 分解成功。如果 info = -i，表示第 i 个参数错误(不包括句柄)。

mcsolverDnXgetrs 支持的算法表

mcsolver_ALG_0 or NULL	默认算法

通用 API 有两种不同的类型，dataTypeA 是矩阵 A 的数据类型，computeType 是操作的计算类型。mcsolverDnXgeqrf 仅支持以下四种组合。

数据类型和计算类型的有效组合

DataTypeA	ComputeType	Meaning
MACA_R_32F	MACA_R_32F	SGETRF
MACA_R_64F	MACA_R_64F	DGETRF
MACA_C_32F	MACA_C_32F	CGETRF
MACA_C_64F	MACA_C_64F	ZGETRF

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。
MCSOLVER_STATUS_NOT_INITIALIZED	库未初始化。
MCSOLVER_STATUS_INVALID_VALUE	传递的参数无效 (n<0 或 lda<max(1,n))。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.4.6 mcsolverDnXsytrs()

下面的辅助函数可以计算预分配缓冲区所需的大小。

```
mcsolverStatus_t
mcsolverDnXsytrs_bufferSize(
    mcsolverDnHandle_t handle,
    mcblasFillMode_t uplo,
    int64_t n,
    int64_t nrhs,
    macaDataType dataTypeA,
    const void *A,
    int64_t lda,
    const int64_t *ipiv,
    macaDataType dataTypeB,
    void *B,
    int64_t ldb,
    size_t *workspaceInBytesOnDevice,
    size_t *workspaceInBytesOnHost);
```

例程如下：

```
mcsolverStatus_t mcsolverAPI mcsolverDnXsytrs(
    mcsolverDnHandle_t handle,
    mcblasFillMode_t uplo,
    int64_t n,
    int64_t nrhs,
    macaDataType dataTypeA,
    const void *A,
    int64_t lda,
    const int64_t *ipiv,
    macaDataType dataTypeB,
    void *B,
    int64_t ldb,
    void *bufferOnDevice,
    size_t workspaceInBytesOnDevice,
    void *bufferOnHost,
    size_t workspaceInBytesOnHost,
    int *info);
```

使用通用 API 接口解决线性方程组。

A 包含了 mcsolverDnXsytrf() 的分解结果，只有下三角部分或上三角部分是有意义的，其他部分保持不变。

如果输入参数 uplo 为 MCBLAS_FILL_MODE_LOWER，则因式分解的细节存储如下：

$$A = L * D * L^T$$

如果输入参数 uplo 为 MCBLAS_FILL_MODE_UPPER，则因式分解的细节存储如下：

$$A = U * D * U^T$$

用户需要提供由 mcsolverDnXsytrf() 获得的主元索引，以及由输入参数 bufferOnDevice 和 bufferOnHost 指向的设备和主机工作空间。输入参数 workspaceInBytesOnDevice 和 workspaceInBytesOnHost 是设备和主机工作空间的字节大小，它们由 mcsolverDnXsytrs_bufferSize() 返回。

如果输出参数 Info = -i （小于零），则第 i 个参数是错误的（不包含句柄）。通用 API 有两种不同的类型，dataTypeA 是矩阵 A 的数据类型，computeType 是操作的计算类型。mcsolverDnXsytrs 仅支持以下四种组合。

mcsolverDnXsytrs_bufferSize 和 mcsolverDnXsytrs 的输入参数列表：

API of sytrs

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库上下文的句柄。
uplo	host	input	指出存储矩阵 A 的下三角还是上三角。另一部分没有引用。
n	host	input	矩阵 A 的列数。
nrhs	host	input	右侧项的数量。
dataTypeA	host	in	数组 A 的数据类型。
A	device	in/out	维度为 lda * n 的数组，其中 lda 不小于 max(1, m)。
lda	host	input	用于存储矩阵 A 的二维数组的前导维度大小。
ipiv	device	input	大小至少为 n 的数组，包含主元索引。
dataTypeB	host	in	数组 B 的数据类型。
B	device	in/out	维度为 ldb * nrhs 的数组，其中 ldb 不小于 max(1,nrhs)。
ldb	host	input	用于存储矩阵 B 的二维数组的前导维度大小。
bufferOnDevice	device	in/out	设备工作空间。类型为 void、大小为 workspaceInBytesOnDevice 字节的数组。
workspaceInBytesOnDevice	host	input	由 mcsolverDnXpotrf_bufferSize 返回的 bufferOnDevice 的字节大小。
bufferOnHost	host	in/out	主机工作空间。类型为 void、大小为 workspaceInBytesOnHost 字节的数组。
workspaceInBytesOnHost	host	input	由 mcsolverDnXpotrf_bufferSize 返回的 bufferOnHost 的字节大小。
info	device	output	如果 info = 0，则 Cholesky 分解成功。如果 info = -i，表示第 i 个参数错误(不包括句柄)。

通用 API 有两种不同的类型：dataTypeA 是矩阵 A 的数据类型，dataTypeB 是矩阵 A 的数据类型。mcsolverDnXsytrs 只支持以下四种组合：

数据类型和计算类型的有效组合

DataTypeA	DataTypeB	Meaning
MACA_R_32F	MACA_R_32F	SSYTRS
MACA_R_64F	MACA_R_64F	DSYTRS
MACA_C_32F	MACA_C_32F	CSYTRS
MACA_C_64F	MACA_C_64F	ZSYTRS

返回状态

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。
MCSOLVER_STATUS_NOT_INITIALIZED	库未初始化。
MCSOLVER_STATUS_INVALID_VALUE	传递的参数无效 (n<0 或 lda<max(1,n))。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.4.7 mcsolverDnXtrtri()

下面的辅助函数可以计算所需的预分配缓冲区大小。

```
mcsolverStatus_t
 mcsolverDnXtrtri_bufferSize(
       mcsolverDnHandle_t handle,
       mcblasFillMode_t uplo,
       mcblasDiagType_t diag,
       int64_t n,
       macaDataType dataTypeA,
       void *A,
       int64_t lda,
       size_t *workspaceInBytesOnDevice,
       size_t *workspaceInBytesOnHost);
```

下面的例程:

```
mcsolverStatus_t
 mcsolverDnXtrtri(
       mcsolverDnHandle_t handle,
       mcblasFillMode_t uplo,
       mcblasDiagType_t diag,
       int64_t n,
       macaDataType dataTypeA,
       void *A,
       int64_t lda,
       void *bufferOnDevice,
       size_t workspaceInBytesOnDevice,
       void *bufferOnHost,
       size_t workspaceInBytesOnHost,
       int *info);
```

使用通用的 API 接口计算三角矩阵的逆。

A 是一个 n×n 的三角矩阵，只有下三角或上三角部分是有效的。输入参数 uplo 表示使用矩阵的哪个部分。函数会保持矩阵的其他部分不变。

如果输入参数 uplo 是 MCBLAS_FILL_MODE_LOWER，则只处理 A 的下三角部分，并将其替换为下三角的逆矩阵。

如果输入参数 uplo 是 MCBLAS_FILL_MODE_UPPER，则只处理 A 的上三角部分，并将其替换为上三角的逆矩阵。

用户需要提供设备和主机工作空间，这些空间由输入参数 bufferOnDevice 和 bufferOnHost 指向。输入参数 workspaceInBytesOnDevice 和 workspaceInBytesOnHost 是设备和主机工作空间的字节大小，它们由 mcsolverDnXtrtri_bufferSize() 返回。

如果矩阵求逆失败，输出参数 info = i 表示 A(i,i) = 0。

如果输出参数 info = -i (小于零)，则第 i 个参数错误（不包含句柄）。

mcsolverDnXtrtri_bufferSize 和 mcsolverDnXtrtri 的输入参数列表如下：

API of trtri

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库的上下文句柄。
uplo	host	input	指出存储矩阵 A 的下三角还是上三角。另一部分没有引用。
diag	host	input	枚举的单位对角线类型。
n	host	input	矩阵 A 的行数和列数。
dataTypeA	host	in	数组 A 的数据类型。

下页继续

表 2.37 – 续上页

参数	内存	In/out	含义
A	device	in/out	维度为 lda * n 的数组，其中 lda 不小于 max(1,n) 。
lda	host	input	用于存储矩阵 A 的二维数组前导维度。
bufferOnDevice	device	in/out	设备工作空间，以 void 类型的数组表示，大小为 workspaceInBytesOnDevice 字节。
workspaceInBytesOnDevice	host	input	bufferOnDevice 的字节大小，由 mcsolverDnXtrtri_bufferSize 返回。
bufferOnHost	host	in/out	主机工作空间，以 void 类型的数组表示，大小为 workspaceInBytesOnHost 字节。
workspaceInBytesOnHost	host	input	bufferOnHost 的字节大小，由 mcsolverDnXtrtri_bufferSize 返回。
info	device	output	如果 info = 0，表示矩阵求逆成功。如果 info = -i，表示第 i 个参数错误（不包含句柄）。如果 info = i，A(i,i) = 0 。

有效数据类型

DataTypeA	Meaning
MACA_R_32F	STRTRI
MACA_R_64F	DTRTRI
MACA_C_32F	CTRTRI
MACA_C_64F	ZTRTRI

返回状态

MCSOLVER_STATUS_SUCCESS	操作成功完成。
MCSOLVER_STATUS_NOT_INITIALIZED	库未进行初始化。
MCSOLVER_STATUS_NOT_SUPPORTED	不支持的数据类型。
MCSOLVER_STATUS_INVALID_VALUE	传递的参数无效（n<0 或 lda<max(1,n)）。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.5 稠密特征值求解器参考（64 位 API）

本节介绍 mcsolverDN 的特征值求解 API，包括双对角化和 SVD。

2.4.5.1 mcsolverDnXgesvd()

下面的辅助函数可以计算所需的预分配缓冲区大小。

```
mcsolverStatus_t
mcsolverDnXgesvd_bufferSize(
    mcsolverDnHandle_t handle,
    mcsolverDnParams_t params,
    signed char jobu,
    signed char jobvt,
    int64_t m,
```

(下页继续)

(续上页)

```
    int64_t n,
    macaDataType dataTypeA,
    const void *A,
    int64_t lda,
    macaDataType dataTypeS,
    const void *S,
    macaDataType dataTypeU,
    const void *U,
    int64_t ldu,
    macaDataType dataTypeVT,
    const void *VT,
    int64_t ldvt,
    macaDataType computeType,
    size_t *workspaceInBytesOnDevice,
    size_t *workspaceInBytesOnHost)
```

例程如下：

```
mcsolverStatus_t
mcsolverDnXgesvd(
    mcsolverDnHandle_t handle,
    mcsolverDnParams_t params,
    signed char jobu,
    signed char jobvt,
    int64_t m,
    int64_t n,
    macaDataType dataTypeA,
    void *A,
    int64_t lda,
    macaDataType dataTypeS,
    void *S,
    macaDataType dataTypeU,
    void *U,
    int64_t ldu,
    macaDataType dataTypeVT,
    void *VT,
    int64_t ldvt,
    macaDataType computeType,
    void *bufferOnDevice,
    size_t workspaceInBytesOnDevice,
    void *bufferOnHost,
    size_t workspaceInBytesOnHost,
    int *info)
```

此函数计算 m×n 矩阵 A 的奇异值分解（SVD）以及对应的左奇异和/或右奇异向量。SVD 的写法如下

$$A = U * \Sigma * V^H$$

其中，Σ 是一个 m×n 的矩阵，除了其 min(m,n) 个对角元素外，其余元素均为零；U 是一个 m×m 的酉矩阵，V 是一个 n×n 的酉矩阵。Σ 的对角元素是矩阵 A 的奇异值；它们是实数非负数，并且按降序返回。矩阵 U 和 V 的前 min(m,n) 列是矩阵 A 的左奇异向量和右奇异向量。

用户必须提供由输入参数 bufferOnDevice and bufferOnHost 指向的设备和主机工作空间。输入参数 workspaceInBytesOnDevice （和 workspaceInBytesOnHost ）是设备（和主机）工作空间的字节大小，可以通过 mcsolverDnXgesvd_bufferSize() 返回。

如果输出参数 Info = -i （小于零），则第 i 个参数是错误的 (不包含句柄)。如果 bdsqr 未收敛，info 指定了中间双对角形式的多少个超对角线未收敛为零。目前，mcsolverDnXgesvd 只支持默认算法。

mcsolverDnXpotrs 支持的算法表

mcsolver_ALG_0 or NULL	默认算法

备注 1：gesvd 仅支持 m>=n 。

备注 2：该例程返回的是 V^H，而不是 V 。

mcsolverDnXgesvd_bufferSize 和 mcsolverDnXgesvd 的输入参数列表：

API of mcsolverDnXgesvd

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库的上下文句柄。
params	host	input	通过 mcsolverDnSetAdvOptions 函数收集的信息所构建的结构体。
jobu	host	input	指定计算矩阵 U 的全部或部分的选项：值为 A 时，返回矩阵 U 的所有 m 列，存储在数组 U 中。值为 S 时，返回矩阵 U 的前 min(m,n) 列（即左奇异向量），存储在数组 U 中。值为 O 时，将矩阵 U 的前 min(m,n) 列（即左奇异向量）覆盖到数组 A 中。值为 N 时，不计算矩阵 U 的任何列（即没有左奇异向量）。
jobvt	host	input	指定计算矩阵 V**T 的全部或部分的选项：值为 A 时，返回矩阵 V**T 的所有 n 行，存储在数组 VT 中。值为 S 时，返回矩阵 V**T 的前 min(m,n) 行（即右奇异向量），存储在数组 VT 中。值为 O 时，将矩阵 V**T 的前 min(m,n) 行（即右奇异向量）覆盖到数组 A 中。值为 N 时，不计算矩阵 V**T 的任何行（即没有右奇异向量）。
m	host	input	矩阵 A 的行数。
n	host	input	矩阵 A 的列数。
dataTypeA	host	input	数组 A 的数据类型。
A	device	in/out	维度为 lda * n 的数组，其中 lda 不小于 max(1,m) 。在退出时，A 的内容将被清空。
lda	host	input	用于存储矩阵 A 的二维数组前导维度。
dataTypeS	host	input	数组 S 的数据类型。
S	device	output	维度为 min(m,n) 的实数数组。A 的奇异值按照 S(i) >= S(i+1) 的顺序排序。
dataTypeU	host	input	数组 U 的数据类型。
U	device	output	维度为 ldu * m 的数组，其中 ldu 不小于 max(1,m)。U 包含大小为 m×m 的酉矩阵 U 。
ldu	host	input	用于存储矩阵 U 的二维数组前导维度。
dataTypeVT	host	input	数组 VT 的数据类型。
VT	device	output	维度为 ldvt * n 的数组，其中 ldvt 不小于 max(1,n) 。VT 包含大小为 n×n 的酉矩阵 V**T 。
ldvt	host	input	用于存储矩阵 Vt 的二维数组前导维度。
computeType	host	input	计算的数据类型。
bufferOnDevice	device	in/out	设备工作空间，以 void 类型的数组表示，大小为 workspaceInBytesOnDevice 字节。

下页继续

表 2.38 – 续上页

参数	内存	In/out	含义
workspaceInBytesOnDevice	host	input	bufferOnDevice 的字节大小，由 mcsolverDnXpotrf_bufferSize 返回。
bufferOnHost	host	in/out	主机工作空间，以 void 类型的数组表示，大小为 workspaceInBytesOnHost 字节。
workspaceInBytesOnHost	host	input	bufferOnHost 的字节大小，由 mcsolverDnXpotrf_bufferSize 返回。
info	device	output	如果 info = 0，表示操作成功。如果 info = -i，表示第 i 个参数错误（不包含句柄）。如果 info > 0，info 表示未收敛为零的中间双对角形式的超对角线数量。

通用 API 有三种不同类型，dataTypeA 是矩阵 A 的数据类型，dataTypeS 是向量 S 的数据类型，dataTypeU 是矩阵 U 的数据类型，dataTypeVT 是矩阵 VT 的数据类型，computeType 是操作的计算类型。mcsolverDnXgesvd 仅支持以下四种组合。

数据类型和计算类型的有效组合

DataTypeA	ComputeType	Meaning
MACA_R_32F	MACA_R_32F	SGETRF
MACA_R_64F	MACA_R_64F	DGETRF
MACA_C_32F	MACA_C_32F	CGETRF
MACA_C_64F	MACA_C_64F	ZGETRF

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。
MCSOLVER_STATUS_NOT_INITIALIZED	库未初始化。
MCSOLVER_STATUS_INVALID_VALUE	传递的参数无效 (n<0 或 lda<max(1,n))。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.5.2 mcsolverDnXgesvdp()

下面的辅助函数可以计算所需的预分配缓冲区大小。

```
mcsolverStatus_t
 mcsolverDnXgesvdp_bufferSize(
       mcsolverDnHandle_t handle,
       mcsolverDnParams_t params,
       mcsolverEigMode_t jobz,
       int econ,
       int64_t m,
       int64_t n,
       macaDataType dataTypeA,
       const void *A,
       int64_t lda,
       macaDataType dataTypeS,
       const void *S,
       macaDataType dataTypeU,
       const void *U,
       int64_t ldu,
```

(下页继续)

(续上页)

```
        macaDataType dataTypeV,
        const void *V,
        int64_t ldv,
        macaDataType computeType,
        size_t *workspaceInBytesOnDevice,
        size_t *workspaceInBytesOnHost)
```

下面的例程:

```
mcsolverStatus_t
 mcsolverDnXgesvdp(
        mcsolverDnHandle_t handle,
        mcsolverDnParams_t params,
        mcsolverEigMode_t jobz,
        int econ,
        int64_t m,
        int64_t n,
        macaDataType dataTypeA,
        void *A,
        int64_t lda,
        macaDataType dataTypeS,
        void *S,
        macaDataType dataTypeU,
        void *U,
        int64_t ldu,
        macaDataType dataTypeV,
        void *V,
        int64_t ldv,
        macaDataType computeType,
        void *bufferOnDevice,
        size_t workspaceInBytesOnDevice,
        void *bufferOnHost,
        size_t workspaceInBytesOnHost,
        int *d_info,
        double *h_err_sigma)
```

此函数计算 m×n 矩阵 A 的奇异值分解（SVD）以及对应的左奇异和/或右奇异向量。SVD 的写法如下：

$$A = U * \Sigma * V^H$$

其中 Σ 是一个 m×n 矩阵，除了其 min(m,n) 个对角元素外，其余元素均为零。U 是一个 m×m 的酉矩阵，V 是一个 n×n 的酉矩阵。Σ 的对角线元素是 A 的奇异值；它们是实数非负数，并以降序返回。U 和 V 的前 min(m,n) 列是 A 的左奇异向量和右奇异向量。

mcsolverDnXgesvdp 结合 [14] 中的极分解和 mcsolverDnXsyevd 来计算 SVD。它比基于 QR 算法的 mcsolverDnXgesvd 更快。然而当矩阵 A 的奇异值接近零时，[14] 中的极分解可能无法得到完全的酉矩阵。为了解决奇异值接近零时的问题，我们添加了一个小的扰动，使极分解能够得到正确的结果。因此奇异值的精度会受到这个扰动的影响。输出参数 h_err_sigma 表示了这个扰动的大小。换句话说，h_err_sigma 反映了 SVD 的精度。

用户需要提供设备和主机工作空间，这些空间由输入参数 bufferOnDevice 和 bufferOnHost 指向。输入参数 workspaceInBytesOnDevice （和 workspaceInBytesOnHost ）是设备（和主机）工作空间的字节大小，它们由 mcsolverDnXgesvdp_bufferSize() 返回。

如果输出参数 info = -i (小于零)，则第 i 个参数错误（不包含句柄）。

目前，mcsolverDnXgesvdp 仅支持默认算法。

mcsolverDnXgesvdp 支持的算法表

MCSOLVER_ALG_0 or NULL	默认算法。

备注 1: gesvdp 也支持 n>=m 的情况。

备注 2: 该例程返回 V,

mcsolverDnXgesvdp_bufferSize 和 mcsolverDnXgesvdp 的输入参数表：

API of mcsolverDnXgesvdp

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库的上下文句柄。
params	host	input	通过 mcsolverDnSetAdvOptions 函数收集的信息所构建的结构体。
jobz	host	input	用于指定计算奇异值还是同时计算奇异向量的选项: jobz = MCSOLVER_EIG_MODE_NOVECTOR : 仅计算奇异值。 jobz = MCSOLVER_EIG_MODE_VECTOR：计算奇异值和奇异向量。
econ	host	input	econ = 1 时，使用经济规模（economy size）计算 U 和 V。
m	host	input	矩阵 A 的行数。
n	host	input	矩阵 A 的列数。
dataTypeA	host	input	数组 A 的数据类型。
A	device	in/out	维度为 lda * n 的数组，其中 lda 不小于 max(1,m) 。在退出时，A 的内容将被清空。
lda	host	input	用于存储矩阵 A 的二维数组前导维度。
dataTypeS	host	input	数组 S 的数据类型。
S	device	output	维度为 min(m,n) 的实数数组，A 的奇异值按照 S(i) >= S(i+1) 的顺序排序。
dataTypeU	host	input	数组 U 的数据类型。
U	device	output	维度为 ldu * m 的数组，其中 ldu 不小于 max(1,m) 。U 包含大小为 m×m 的酉矩阵 U。如果 econ=1 ，仅返回 U 的前 min(m, n) 列。
ldu	host	input	用于存储矩阵 U 的二维数组前导维度。
dataTypeV	host	input	数组 V 的数据类型。
V	device	output	维度为 ldv * n 的数组，其中 ldv 不小于 max(1,n) 。V 包含大小为 n×n 的酉矩阵 V。如果 econ=1 ，仅返回 V 的前 min(m, n) 列。
ldv	host	input	用于存储矩阵 V 的二维数组前导维度。
computeType	host	input	计算的数据类型。
bufferOnDevice	device	in/out	设备工作空间，以 void 类型的数组表示，大小为 workspaceInBytesOnDevice 字节。
workspaceInBytesOnDevice	host	input	bufferOnDevice 的字节大小，由 mcsolverDnXpotrf_bufferSize 返回。
bufferOnHost	host	in/out	主机工作空间，以 void 类型的数组表示，大小为 workspaceInBytesOnHost 字节。
workspaceInBytesOnHost	host	input	bufferOnHost 的字节大小，由 mcsolverDnXpotrf_bufferSize 返回。

下页继续

表 2.39 – 续上页

参数	内存	In/out	含义
info	device	output	如果 info = 0，表示操作成功。如果 info = -i，表示第 i 个参数错误（不包含句柄）。
h_err_sigma	host	output	扰动的大小，表示奇异值分解的准确性。

通用 API 有两种不同的类型。dataTypeA 是矩阵 A 的数据类型，dataTypeS 是向量 S 的数据类型，dataTypeU 是矩阵 U 的数据类型。dataTypeV 是矩阵 V 的数据类型。computeType 是操作的计算类型。mcsolverDnXgesvdp 仅支持以下四种组合：

数据类型和计算类型的有效组合

DataTypeA	DataTypeS	DataTypeU	DataTypeV	ComputeType	Meaning
MACA_R_32F	MACA_R_32F	MACA_R_32F	MACA_R_32F	MACA_R_32F	SGESVDP
MACA_R_64F	MACA_R_64F	MACA_R_64F	MACA_R_64F	MACA_R_64F	DGESVDP
MACA_C_32F	MACA_R_32F	MACA_C_32F	MACA_C_32F	MACA_C_32F	CGESVDP
MACA_C_64F	MACA_R_64F	MACA_C_64F	MACA_C_64F	MACA_C_64F	ZGESVDP

返回状态

MCSOLVER_STATUS_SUCCESS	操作成功完成。
MCSOLVER_STATUS_NOT_INITIALIZED	库未进行初始化。
MCSOLVER_STATUS_INVALID_VALUE	传递的参数无效 (m,n<0 或 lda<max(1,m) 或 ldu<max(1,m) 或 ldv<max(1,n))。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.5.3 mcsolverDnXgesvdr()

下面的辅助函数可以计算所需的预分配缓冲区大小。

```
mcsolverStatus_t
 mcsolverDnXgesvdr_bufferSize (
        mcsolverDnHandle_t handle,
        mcsolverDnParams_t params,
        signed char jobu,
        signed char jobv,
        int64_t m,
        int64_t n,
        int64_t k,
        int64_t p,
        int64_t niters,
        macaDataType dataTypeA,
        const void *A,
        int64_t lda,
        macaDataType dataTypeSrand,
        const void *Srand,
        macaDataType dataTypeUrand,
        const void *Urand,
        int64_t ldUrand,
        macaDataType dataTypeVrand,
        const void *Vrand,
        int64_t ldVrand,
        macaDataType computeType,
        size_t *workspaceInBytesOnDevice,
        size_t *workspaceInBytesOnHost )
```

下面的例程

```
mcsolverStatus_t
 mcsolverDnXgesvdr(
       mcsolverDnHandle_t handle,
       mcsolverDnParams_t params,
       signed char jobu,
       signed char jobv,
       int64_t m,
       int64_t n,
       int64_t k,
       int64_t p,
       int64_t niters,
       macaDataType dataTypeA,
       void *A,
       int64_t lda,
       macaDataType dataTypeSrand,
       void *Srand,
       macaDataType  dataTypeUrand,
       void *Urand,
       int64_t ldUrand,
       macaDataType dataTypeVrand,
       void *Vrand,
       int64_t ldVrand,
       macaDataType computeType,
       void *bufferOnDevice,
       size_t workspaceInBytesOnDevice,
       void *bufferOnHost,
       size_t workspaceInBytesOnHost,
       int *d_info)
```

该函数计算一个 m×n 矩阵 A 的近似秩-k 奇异值分解 (k-SVD)，以及相应的左奇异向量和/或右奇异向量。k-SVD 可以表示为:

$$A_k \approx U * \Sigma * V^H$$

其中 Σ 是一个 k×k 矩阵，除了对角线元素外都为零。U 是一个 m×k 正交矩阵，V 是一个 k×n 正交矩阵。Σ 的对角线元素是 A 的近似奇异值；它们是实数且非负，并以降序返回。U 和 V 的列向量是 A 的前 k 个左奇异向量和右奇异向量。

mcsolverDnXgesvdr 实现了 [15] 中描述的随机方法来计算 k-SVD，如果满足 [15] 中描述的条件，则可以高概率地得到准确的结果。mcsolverDnXgesvdr 用于快速高质量地计算 A 的一小部分特征值 (即 k 远小于 min(m,n))，尤其适用于矩阵维度较大的情况。

算法的精度取决于 A 的特征值谱、迭代次数 niters 、过采样参数 p 以及 p 和矩阵 A 维度之间的比例。较大的过采样参数 p 或更多的迭代次数 niters 可以产生更准确的近似结果，但同时也会增加 mcsolverDnXgesvdr 的运行时间。

我们建议使用两次迭代，并将过采样设置为至少 2k。一旦求解器提供足够的精度，可以调整 k 和 niters 的值以获得更好的性能。

用户需要提供设备和主机工作空间，这些空间由输入参数 bufferOnDevice and bufferOnHost 指向。输入参数 workspaceInBytesOnDevice (和 workspaceInBytesOnHost) 是设备 (和主机) 工作空间的字节大小，它们由 mcsolverDnXgesvdr_bufferSize() 返回。

如果输出参数 info = -i (小于零)，则第 i 个参数错误（不包含句柄）。

目前，mcsolverDnXgesvdr 仅支持默认算法。

mcsolverDnXgesvdr 支持的算法表

MCSOLVER_ALG_0 or NULL	默认算法。

备注 1: gesvdr 也支持 n>=m 的情况。

备注 2: 该例程返回 V

mcsolverDnXgesvdr_bufferSize 和 mcsolverDnXgesvdr 的输入参数表：

API of mcsolverDnXgesvdr

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库的上下文句柄。
params	host	input	通过 mcsolverDnSetAdvOptions 函数收集的信息所构建的结构体。
jobu	host	input	指定计算矩阵 U 的全部或部分的选项。值为 S 时，将前 k 列的 U（左奇异向量）返回到数组 U 中。值为 N 时，不计算任何 U 列（无左奇异向量）。
jobv	host	input	指定计算矩阵 V 的全部或部分的选项。值为 S 时，将前 k 行的 V（右奇异向量）返回到数组 V 中。值为 N 时，不计算任何 V 行（无右奇异向量）。
m	host	input	矩阵 A 的行数。
n	host	input	矩阵 A 的列数。
k	host	input	矩阵 A 的 k-SVD 分解的秩。rank 小于 min(m,n)。
p	host	input	过采样。子空间的大小将为 (k + p)。(k+p) 小于 min(m,n)。
niters	host	input	幂方法的迭代次数。
dataTypeA	host	input	数组 A 的数据类型。
A	device	in/out	维度为 lda * n 的数组，其中 lda 不小于 max(1,m)。在退出时，A 的内容将被清空。
lda	host	input	用于存储矩阵 A 的二维数组前导维度。
dataTypeS	host	input	数组 S 的数据类型。
S	device	output	维度为 min(m,n) 的实数数组，A 的奇异值按照 S(i) >= S(i+1) 的顺序排序。
dataTypeU	host	input	数组 U 的数据类型。
U	device	output	维度为 ldu * m 的数组，其中 ldu 不小于 max(1,m)。U 包含大小为 m×m 的酉矩阵 U。如果 jobu=S，仅返回 U 的前 min(m,n) 列。
ldu	host	input	用于存储矩阵 U 的二维数组前导维度。
dataTypeV	host	input	数组 V 的数据类型。
V	device	output	维度为 ldv * n 的数组，其中 ldv 不小于 max(1,n)。V 包含大小为 n×n 的酉矩阵 V。如果 jobv=S，仅返回 V 的前 min(m,n) 列。
ldv	host	input	用于存储矩阵 V 的二维数组前导维度。
computeType	host	input	计算的数据类型。
bufferOnDevice	device	in/out	设备工作空间，以 void 类型的数组表示，大小为 workspaceInBytesOnDevice 字节。
workspaceInBytesOnDevice	host	input	bufferOnDevice 的字节大小，由 mcsolverDnXgesvdr_bufferSize 返回。
bufferOnHost	host	in/out	主机工作空间，以 void 类型的数组表示，大小为 workspaceInBytesOnHost 字节。

下页继续

表 2.40 – 续上页

参数	内存	In/out	含义
workspaceInBytesOnHost	host	input	bufferOnHost 的字节大小，由 mcsolverDnXgesvdr_bufferSize 返回。
d_info	device	output	如果 info = 0，表示操作成功。如果 info = -i，表示第 i 个参数错误（不包含句柄）。

通用 API 有两种不同的类型。dataTypeA 是矩阵 A 的数据类型，dataTypeS 是向量 S 的数据类型，dataTypeU 是矩阵 U 的数据类型。dataTypeV 是矩阵 V 的数据类型。computeType 是操作的计算类型。mcsolverDnXgesvdr 仅支持以下四种组合：

数据类型和计算类型的有效组合

DataTypeA	DataTypeS	DataTypeU	DataTypeV	ComputeType	Meaning
MACA_R_32F	MACA_R_32F	MACA_R_32F	MACA_R_32F	MACA_R_32F	SGESVDR
MACA_R_64F	MACA_R_64F	MACA_R_64F	MACA_R_64F	MACA_R_64F	DGESVDR
MACA_C_32F	MACA_R_32F	MACA_C_32F	MACA_C_32F	MACA_C_32F	CGESVDR
MACA_C_64F	MACA_R_64F	MACA_C_64F	MACA_C_64F	MACA_C_64F	ZGESVDR

返回状态

MCSOLVER_STATUS_SUCCESS	操作成功完成。
MCSOLVER_STATUS_NOT_INITIALIZED	库未进行初始化。
MCSOLVER_STATUS_INVALID_VALUE	传递的参数无效 (m,n<0 或 lda<max(1,m) 或 ldu<max(1,m) 或 ldv<max(1,n))。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.5.4 mcsolverDnXsyevd()

下面的辅助函数可以计算所需的预分配缓冲区大小。

```
mcsolverStatus_t
mcsolverDnXsyevd_bufferSize(
    mcsolverDnHandle_t handle,
    mcsolverDnParams_t params,
    mcsolverEigMode_t jobz,
    mcblasFillMode_t uplo,
    int64_t n,
    macaDataType dataTypeA,
    const void *A,
    int64_t lda,
    macaDataType dataTypeW,
    const void *W,
    macaDataType computeType,
    size_t *workspaceInBytesOnDevice,
    size_t *workspaceInBytesOnHost)
```

例程如下：

```
mcsolverStatus_t
mcsolverDnXsyevd(
    mcsolverDnHandle_t handle,
    mcsolverDnParams_t params,
```

(下页继续)

(续上页)

```
    mcsolverEigMode_t jobz,
    mcblasFillMode_t uplo,
    int64_t n,
    macaDataType dataTypeA,
    void *A,
    int64_t lda,
    macaDataType dataTypeW,
    void *W,
    macaDataType computeType,
    void *bufferOnDevice,
    size_t workspaceInBytesOnDevice,
    void *bufferOnHost,
    size_t workspaceInBytesOnHost,
    int *info)
```

使用通用 API 计算对称 (厄米特) n×n 矩阵 A 的特征值和特征向量。标准对称特征值问题是

$$A * V = V * \Lambda$$

其中 Λ 是一个 n×n 实对角矩阵。V 是一个 n×n 的酉矩阵。Λ 的对角线元素是 A 的特征值升序排列。

用户必须提供由输入参数 bufferOnDevice and bufferOnHost 指向的设备和主机工作空间。输入参数 workspaceInBytesOnDevice （和 workspaceInBytesOnHost ）是设备（和主机）工作空间的字节大小，并且可以通过 mcsolverDnXsyevd_bufferSize() 返回。

如果输出参数 Info = -i （小于零），则第 i 个参数是错误的（不包含句柄）。如果 info = i （大于零），则中间三对角形式的 i 个非对角元素未收敛为零。

如果 jobz = mcsolver_EIG_MODE_VECTOR，A 包含矩阵 A 的正交特征向量。特征向量由分治算法算法计算。目前，mcsolverdnxsyyevd 只支持默认算法。

mcsolverDnXpotrs 支持的算法表

mcsolver_ALG_0 or NULL	默认算法

mcsolverDnXsyevd_bufferSize 和 mcsolverDnXsyevd 输入参数表：

API of mcsolverDnXsyevd

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库的上下文句柄。
params	host	input	通过 mcsolverDnSetAdvOptions 函数收集的信息所构建的结构体。
jobz	host	input	指定计算特征值或同时计算特征值和特征向量的选项：jobz = MCSOLVER_EIG_MODE_NOVECTOR：仅计算特征值；jobz = MCSOLVER_EIG_MODE_VECTOR：计算特征值和特征向量。
uplo	host	input	指定存储的矩阵 A 的部分：uplo = MCBLAS_FILL_MODE_LOWER：存储 A 的下三角部分。uplo = MCBLAS_FILL_MODE_UPPER：存储 A 的上三角部分。
n	host	input	矩阵 A 的行数（或列数）。
dataTypeA	host	in	数组 A 的数据类型。

下页继续

表 2.41 – 续上页

参数	内存	In/out	含义
A	device	in/out	维度为 lda * n 的数组，其中 lda 不小于 max(1,n) 。如果 uplo = MCBLAS_FILL_MODE_UPPER ，则 A 的前导的 n * n 的上三角部分包含了矩阵 A 的上三角部分。如果 uplo = MCBLAS_FILL_MODE_LOWER ，则 A 的前导的 n * n 的下三角部分包含了矩阵 A 的下三角部分。在退出时，如果 jobz = MCSOLVER_EIG_MODE_VECTOR 且 info = 0，则 A 包含矩阵 A 的正交特征向量。如果 jobz = MCSOLVER_EIG_MODE_NOVECTOR ，则 A 的内容将被清空。
lda	host	input	用于存储矩阵 A 的二维数组前导维度。
dataTypeW	host	in	数组 W 的数据类型。
W	device	output	维度为 n 的实数数组。包含矩阵 A 的特征值，按升序排列，即满足 W(i) <= W(i+1) 。
computeType	host	in	计算的数据类型。
bufferOnDevice	device	in/out	设备工作空间，以 void 类型的数组表示，大小为 workspaceInBytesOnDevice 字节。
workspaceInBytesOnDevice	host	input	bufferOnDevice 的字节大小，由 mcsolverDnXpotrf_bufferSize 返回。
bufferOnHost	host	in/out	主机工作空间，以 void 类型的数组表示，大小为 workspaceInBytesOnHost 字节。
workspaceInBytesOnHost	host	input	bufferOnHost 的字节大小，由 mcsolverDnXpotrf_bufferSize 返回。
info	device	output	如果 info = 0 ，表示操作成功。如果 info = -i ，表示第 i 个参数错误（不包含句柄）。如果 info = i(>0) ，info 表示中间三对角形式的第 i 个非对角元素未收敛为零。

通用 API 有三种不同类型，dataTypeA 是矩阵 A 的数据类型，dataTypeW 是矩阵 W 的数据类型，computeType 是操作的计算类型。mcsolverdnxsyyevd 仅支持以下四种组合。

数据类型和计算类型的有效组合

DataTypeA	DataTypeW	ComputeType	Meaning
MACA_R_32F	MACA_R_32F	MACA_R_32F	SSYEVD
MACA_R_64F	MACA_R_64F	MACA_R_64F	DSYEVD
MACA_C_32F	MACA_R_32F	MACA_C_32F	CHEEVD
MACA_C_64F	MACA_R_64F	MACA_C_64F	ZHEEVD

值	含义
MCSOLVER_STATUS_SUCCESS	初始化成功。
MCSOLVER_STATUS_NOT_INITIALIZED	库未初始化。
MCSOLVER_STATUS_INVALID_VALUE	传递的参数无效 (n<0 或 lda<max(1,n))。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。

2.4.5.5 mcsolverDnXsyevdx()

下面的辅助函数可以计算所需的预分配缓冲区大小。

```
mcsolverStatus_t
 mcsolverDnXsyevdx_bufferSize(
     mcsolverDnHandle_t handle,
     mcsolverDnParams_t params,
     mcsolverEigMode_t jobz,
     mcsolverEigRange_t range,
     mcblasFillMode_t uplo,
     int64_t n,
     macaDataType dataTypeA,
     const void *A,
     int64_t lda,
     void *vl,
     void *vu,
     int64_t il,
     int64_t iu,
     int64_t *h_meig,
     macaDataType dataTypeW,
     const void *W,
     macaDataType computeType,
     size_t *workspaceInBytesOnDevice,
     size_t *workspaceInBytesOnHost)
```

下面的例程:

```
mcsolverStatus_t MCSOLVERAPI mcsolverDnXsyevdx(
     mcsolverDnHandle_t handle,
     mcsolverDnParams_t params,
     mcsolverEigMode_t jobz,
     mcsolverEigRange_t range,
     mcblasFillMode_t uplo,
     int64_t n,
     macaDataType dataTypeA,
     void *A,
     int64_t lda,
     void * vl,
     void * vu,
     int64_t il,
     int64_t iu,
     int64_t *meig64,
     macaDataType dataTypeW,
     void *W,
     macaDataType computeType,
     void *bufferOnDevice,
     size_t workspaceInBytesOnDevice,
     void *bufferOnHost,
     size_t workspaceInBytesOnHost,
     int *info)
```

使用通用 API 接口，计算对称（厄米特）n×n 矩阵 A 的所有或部分特征值，以及可选的特征向量。标准的对称特征值问题可以表示为：

$$A * V = V * \Lambda$$

其中 Λ 是一个 n×h_meig 的实数对角矩阵。V 是一个 n×h_meig 的酉矩阵。h_meig 是由该例程计算的特征值/特征向量的数量，当请求整个特征值谱（例如 range = MCSOLVER_EIG_RANGE_ALL ）时，

h_meig 等于 n。Λ 的对角元素是 A 的特征值，按升序排列。

用户需要提供设备和主机工作空间，这些空间由输入参数 bufferOnDevice 和 bufferOnHost 指向。输入参数 workspaceInBytesOnDevice （和 workspaceInBytesOnHost）是设备（和主机）工作空间的字节大小，它们由 mcsolverDnXsyevdx_bufferSize() 返回。

如果输出参数 info = -i (小于零)，则第 i 个参数错误（不包含句柄）。如果 info = i (大于零)，则中间三对角形式的 i 个非对角元素未收敛到零。

如果 jobz = MCSOLVER_EIG_MODE_VECTOR，则 A 包含矩阵 A 的正交特征向量。特征向量是通过分治算法计算得到的。

目前，mcsolverDnXsyevdx 仅支持默认算法。

mcsolverDnXsyevdx 支持的算法表

MCSOLVER_ALG_0 or NULL	默认算法。

mcsolverDnXsyevdx_bufferSize 和 mcsolverDnXsyevdx 的输入参数表：

API of mcsolverDnXsyevdx

参数	内存	In/out	含义
handle	host	input	mcsolverDN 库的上下文句柄。
params	host	input	通过 mcsolverDnSetAdvOptions 函数收集的信息所构建的结构体。
jobz	host	input	指定计算特征值或同时计算特征值和特征向量的选项：jobz = MCSOLVER_EIG_MODE_NOVECTOR：仅计算特征值；jobz = MCSOLVER_EIG_MODE_VECTOR：计算特征值和特征向量。
range	host	input	指定要计算的特征值和可选特征向量的选择范围的选项：range = MCSOLVER_EIG_RANGE_ALL：将找到所有特征值/特征向量，这将变成经典的 syevd / heevd 例程；range = MCSOLVER_EIG_RANGE_V：将找到半开区间 (vl，vu] 内的所有特征值/特征向量；range = MCSOLVER_EIG_RANGE_I：将找到第 il 到第 iu 个特征值/特征向量；
uplo	host	input	指定存储的矩阵 A 的部分。uplo = MCBLAS_FILL_MODE_LOWER：存储 A 的下三角部分。uplo = MCBLAS_FILL_MODE_UPPER：存储 A 的上三角部分。
n	host	input	矩阵 A 的行数（或列数）。
dataTypeA	host	in	数组 A 的数据类型。

下页继续

表 2.42 – 续上页

参数	内存	In/out	含义
A	device	in/out	维度为 lda * n 的数组，其中 lda 不小于 max(1,n) 。如果 uplo = MCBLAS_FILL_MODE_UPPER ，则 A 的前导的 n * n 的上三角部分包含了矩阵 A 的上三角部分。如果 If uplo = MCBLAS_FILL_MODE_LOWER ，则 A 的前导的 n * n 的下三角部分包含了矩阵 A 的下三角部分。在退出时，如果 jobz = MCSOLVER_EIG_MODE_VECTOR 且 info = 0，则 A 包含矩阵 A 的正交特征向量。如果 jobz = MCSOLVER_EIG_MODE_NOVECTOR ，则 A 的内容将被清空。
lda	host	input	用于存储矩阵 A 的二维数组前导维度。lda 不小于 max(1,n) 。
vl,vu	host	input	如果 range = MCSOLVER_EIG_RANGE_V ，则为搜索特征值的区间的下限和上限。vl > vu。如果 range = MCSOLVER_EIG_RANGE_ALL 或 range = MCSOLVER_EIG_RANGE_I ，则不使用该参数。注意，如果特征值非常接近，显然两个不同的特征值例程可能在相同的区间内找到略微不同数量的特征值。这是由于不同的特征值算法，甚至是相同的算法但不同的运行，可能会找到在机器精度附近存在舍入误差的特征值。因此，如果用户希望确保不会错过区间边界内的任何特征值，我们建议用户从区间边界中减去/增加 epsilon (机器精度) ，例如 (vl=vl-eps, vu=vu+eps]。这个建议对于来自 mcSOLVER 或 LAPACK 的任何选择性例程都适用。
il,iu	host	input	整数。如果 range = MCSOLVER_EIG_RANGE_I ，则返回最小和最大特征值的索引（按升序排序）。如果 n > 0，1 <= il <= iu <= n；如果 n = 0，il = 1 且 iu = 0 。如果 range = MCSOLVER_EIG_RANGE_ALL 或 range = MCSOLVER_EIG_RANGE_V ，则不使用该参数。
h_meig	host	output	整数。找到的特征值的总数。0 <= h_meig <= n。如果 range = MCSOLVER_EIG_RANGE_ALL ，h_meig = n。如果 range = MCSOLVER_EIG_RANGE_I ，h_meig = iu-il+1。
dataTypeW	host	in	数组 W 的数据类型。
W	device	output	维度为 n 的实数数组。包含矩阵 A 的特征值，按升序排列，即满足 W(i) <= W(i+1) 。
computeType	host	in	计算的数据类型。
bufferOnDevice	device	in/out	设备工作空间，以 void 类型的数组表示，大小为 workspaceInBytesOnDevice 字节。

下页继续

表 2.42 – 续上页

参数	内存	In/out	含义
workspaceInBytesOnDevice	host	input	bufferOnDevice 的字节大小，由 mcsolverDnXpotrf_bufferSize。
bufferOnHost	host	in/out	主机工作空间，以 void 类型的数组表示，大小为 workspaceInBytesOnHost 字节。
workspaceInBytesOnHost	host	input	bufferOnHost 的字节大小，由 mcsolverDnXpotrf_bufferSize。
info	device	output	如果 info = 0，表示操作成功。如果 info = -i，表示第 i 个参数错误（不包含句柄）。如果 info = i(>0)，info 表示中间三对角形式的第 i 个非对角元素未收敛为零。

通用 API 有三种不同的类型。dataTypeA 是矩阵 A 的数据类型，dataTypeW 是矩阵 W 的数据类型，computeType 是操作的计算类型。mcsolverDnXsyevdx 仅支持以下四种组合：

数据类型和计算类型的有效组合

DataTypeA	DataTypeW	ComputeType	Meaning
MACA_R_32F	MACA_R_32F	MACA_R_32F	SSYEVDX
MACA_R_64F	MACA_R_64F	MACA_R_64F	DSYEVDX
MACA_C_32F	MACA_R_32F	MACA_C_32F	CHEEVDX
MACA_C_64F	MACA_R_64F	MACA_C_64F	ZHEEVDX

返回状态

MCSOLVER_STATUS_SUCCESS	操作成功完成。
MCSOLVER_STATUS_NOT_INITIALIZED	库未进行初始化。
MCSOLVER_STATUS_INVALID_VALUE	传递的参数无效（n<0，或 lda<max(1,n)，或 jobz 不是 MCSOLVER_EIG_MODE_NOVECTOR 或 MCSOLVER_EIG_MODE_VECTOR，或 range 不是 MCSOLVER_EIG_RANGE_ALL 或 MCSOLVER_EIG_RANGE_V 或 MCSOLVER_EIG_RANGE_I，或 uplo 不是 MCBLAS_FILL_MODE_LOWER 或 MCBLAS_FILL_MODE_UPPER）。
MCSOLVER_STATUS_INTERNAL_ERROR	内部操作失败。