曦云® 系列通用计算 GPU

mcFFT API 参考

CSRD-23015-020-F3_V03　　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
V03	2024-03-15	新增曦云®系列 GPU 产品信息
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

本文档介绍了沐曦 MXMACA® 快速傅立叶变换 (FFT) 产品 mcFFT。mcFFT 库旨在为沐曦 GPU 提供高性能支持。

FFT 是一种分治算法，用于高效地计算复数或实数集的离散傅里叶变换。它是计算物理学和通用信号处理中最重要、应用最广泛的数值算法之一。mcFFT 库为在沐曦 GPU 上计算 FFT 提供了一个简单的接口，允许用户可以在高度优化和测试的 FFT 库中快速利用 GPU 的浮点计算能力和并行性。

mcFFT 产品在沐曦 GPU 上广泛且有效支持各种 FFT 输入和选项。该版本的 mcFFT 库支持以下功能：

- 高度优化的算法可以适用于形如 $2^a \times 3^b \times 5^c \times 7^d$ 的输入数据上。通常来说, 素数因子越小，性能越好。即，二次幂最快
- $O(n \log n)$ 算法适用于各种大小的输入数据
- 支持单精度 (32bit 浮点数) 和双精度 (64bit 浮点数) 的 FFT。低精度浮点数的 FFT 拥有更好的性能
- 支持复数和实数的输入和输出。实数输入输出和复数输入输出相比较而言，实数要求的算力更少并且处理速度更快。支持的类型包括：
  - C2C - 复数输入到复数输出的 FFT
  - R2C - 实数输入到复数输出的 FFT
  - C2R - 对称复数输入到实数输出的 FFT
- 支持一维，二维和三维变换
- 支持复数个一维，二维和三维同时变换。和单个变换相比，批处理变换更加高效
- 支持原地变换和异地变换
- 支持任一维度内和维度间的元素步长 (支持跨步布局)
- 支持流式执行。允许异步计算和数据传输

1.1 安装 mcFFT

$$MXMACA \begin{cases} library & \begin{cases} mcFFT \\ \vdots \end{cases} \\ \vdots \\ \vdots \end{cases}$$

mcFFT 是随 MXMACA 工具包一起发布的库。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》。安装完成后，请确保设置了环境变量 MACA_PATH。

```
export MACA_PATH=/your/maca/path
```

mcFFT API 相关文档如下：

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

#lib location:
${MACA_PATH}/lib/libmcfft.so
${MACA_PATH}/lib/libmcfft-device-0.so
${MACA_PATH}/lib/libmcfft-device-1.so
${MACA_PATH}/lib/libmcfft-device-2.so
${MACA_PATH}/lib/libmcfft-device-3.so
```

1.2 Hello mcFFT

以下示例代码演示了使用 mcFFT 库 API 编写的 C 语言应用程序：

```
#CMakeLists.txt
cmake_minimum_required(VERSION 3.16)
project(mcfft_samples)
if(NOT DEFINED ENV{MACA_PATH})
    message(FATAL_ERROR "not defined environment variable: MACA_PATH")
endif()
INCLUDE_DIRECTORIES($ENV{MACA_PATH}/include/)
INCLUDE_DIRECTORIES($ENV{MACA_PATH}/include/mcfft/)
INCLUDE_DIRECTORIES($ENV{MACA_PATH}/mcr/)
LINK_DIRECTORIES($ENV{MACA_PATH}/lib)
set(LINK_LIBS
    mc_runtime
    mcfft
    mcfft-device-0
    mcfft-device-1
    mcfft-device-2
    mcfft-device-3
    )
set(sample_list complex_1d)

foreach(sample ${sample_list})
  add_executable(mcfft_${sample} ${sample}.cpp)
  target_link_libraries(mcfft_${sample} ${LINK_LIBS})
endforeach()
```

```
//complex_1d.cpp, 1D, C2C, in-place
//-------------------------------------------------------------
#include <cassert>
#include <complex>
#include <iostream>
#include <vector>
#include <random>
#include <mc_runtime.h>
#include "mcfft.h"
#define NX 8
#define BATCH 1

int main(int argc, char* argv[])
{
   std::cout << "mcFFT complex 1d FFT example\n";
```

(下页继续)

(续上页)

```
  mcfftHandle plan;
  mcError_t err;

  //在主机处初始化数据：
  std::vector<std::complex<float>> data(NX*BATCH);
  std::vector<std::complex<float>> outData(NX*BATCH);
  std::cout << "Input:\n";

  for(int b = 0; b < BATCH; b++)
  {
     for(int n = 0; b < NX; n++)
     {
        std::mt19937 gen(n);
        const float x = (float)gen() / (float)gen.max();
        const float y = (float)gen() / (float)gen.max();
        const std::complex<float> val(x,y);
        data[NX*b + n] = val;
        std::cout<<data[NX*b + n]<<" ";
     }
  }

  // 创建设备对象和计划：
  mcfftComplex *devPtrData=nullptr;
  err=mcMalloc((void**)&devPtrData, sizeof(mcfftComplex)*NX*BATCH);
  if (err != mcSuccess)
  {
    fprintf(stderr, "Error: Failed to allocate\n");
    return EXIT_FAILURE;
  }

  if (mcfftPlan1d(&plan, NX, MCFFT_C2C, BATCH) != mcSuccess)
  {
    fprintf(stderr, "mcFFT Error: Plan creation failed\n");
    return EXIT_FAILURE;
  }

  //复制数据
  err = mcMemcpy(devPtrData, data.data(), sizeof(mcfftComplex)*NX*BATCH,
↪mcMemcpyHostToDevice);
  if (err != mcSuccess)
  {
    fprintf(stderr, "Error: Failed to copy host to device\n");
    return EXIT_FAILURE;
  }

  // 执行正向变换和原地变换
  if (mcfftExecC2C(plan, devPtrData, devPtrData, MCFFT_FORWARD) !=
↪mcSuccess)
  {
    fprintf(stderr, "mcFFT error: ExecC2C Forward failed\n");
    return EXIT_FAILURE;
  }

  // 执行逆向变换和原地变换
  if (mcfftExecC2C(plan, devPtrData, devPtrData, MCFFT_INVERSE) !=
↪mcSuccess)
```

(下页继续)

(续上页)

```
    {
      fprintf(stderr, "mcFFT error: ExecC2C Inverse failed\n");
      return EXIT_FAILURE;
    }

    /*
    *    在所有任务完成之前，可能无法立即获得结果
    *
    */

    if (mcDeviceSynchronize() != mcSuccess)
    {
      fprintf(stderr, "Error: Failed to synchronize\n");
      return EXIT_FAILURE;
    }

    err = mcMemcpy(outData.data(), devPtrData,
↪sizeof(mcfftComplex)*NX*BATCH, mcMemcpyDeviceToHost);
    if (err != mcSuccess)
    {
      fprintf(stderr, "Error: Failed to copy device to host\n");
      return EXIT_FAILURE;
    }

    const float overN = 1.0f / Nx;
    float       error = 0.0f;
    for(size_t i = 0; i < data.size(); i++)
    {
      float diff = std::max(std::abs(data[i].real() - outData[i].real() *
↪overN),
                            std::abs(data[i].imag() - outData[i].imag() *
↪overN));
      if(diff > error)
      {
        error = diff;
      }
    }

    std::cout << "Transformed back:\n";
    for(size_t i = 0; i < outData.size(); i++)
    {
      std::cout << outData[i]*overN << " ";
    }
    std::cout << std::endl;
    std::cout << "Maximum error: " << error << "\n";

    mcfftDestroy(plan);
    mcFree(devPtrData);
}
```

上述文件 (CMakeList.txt 和 complex_1d.cpp) 在同一个目录中，比如 `/your/example/path`。示例编译步骤如下：

```
$ cd /your/example/path
$ mkdir build && cd build
$ cmake ..
$ make
```

2 mcFFT 使用方法

本章节提供了 mcFFT 库 API 的概览。

离散傅里叶变换 (DFT) 将一个复数向量 x_k (时域) 映射到其频域表示，表示为：

$$X_k = \sum_{n=0}^{N-1} x_n e^{-2\pi i \frac{kn}{N}}$$

X_k 是一个复数向量。时域映射到频域为正向 DFT。如果指数中 e 的符号改为正号，该变换就是逆向变换。根据 N 的大小，会采用不同的算法来获得最佳性能。

mcFFT API 以 FFTW 为模型，后者是基于 CPU，最流行、最高效的的 FFT 库之一。mcFFT 提供了一种简单的配置机制，称为计划 (plan)，该计划使用内部构建块来针对给定配置和选择的特定 GPU 硬件进行优化变换。然后，当调用执行函数时，实际的变换按照计划的执行来进行。这种方法的优势在于，一旦用户创建了一个计划，库将保留执行计划所需的状态，以便多次执行计划而无需重新计算配置。对于 mcFFT 来说，这种模型非常有效，因为不同类型的 FFT 需要不同的线程配置和 GPU 资源，而计划接口提供了一种简单的方式来重用配置。

2.1 傅里叶变换类型

除了一般的复数到复数 (complex-to-complex，C2C) 的变换，mcFFT 还有效地实现了另外两种类型的变换：实数到复数 (real-to-complex，R2C) 的变换与复数到实数 (complex-to-real，C2R) 的变换。在许多实际应用中，输入向量是实数值的。可以很容易地证明，在这种情况下，输出满足厄米特矩阵对称 (Hermitian symmetry)($X_k = X^*_{N-k}$ ，其中星号表示复数共轭)。反之也成立：对于复数域的共轭-厄米 (complex-Hermitian) 输入，逆向变换将是纯实数值。mcFFT 利用了这种冗余，只对厄米特矩阵向量的前半部分起作用。单精度和双精度的变换执行函数被分别定义为:

- `mcfftExecC2C()` / `mcfftExecZ2Z()` - 单/双精度复数到复数的变换。
- `mcfftExecR2C()` / `mcfftExecD2Z()` - 单/双精度实数到复数的正向变换。
- `mcfftExecC2R()` / `mcfftExecZ2D()` - 单/双精度复数到实数的逆向变换。

这些函数中的每一个都需要不同的输入数据布局。

注解: 复数到实数 (C2R) 变换接受复数域的厄米特矩阵输入，这要求第 0 个元素（和第 $\frac{N}{2}$ 个输入，如果 N 为偶数）为实数值，即其虚部应为零。否则，变换的行为是未定义的。

2.2 数据布局

在 mcFFT 库中，数据布局严格依赖于配置和变换类型。在一般复数到复数变换的情况下，输入和输出数据应分别为单精度和双精度模式下的 `mcfftComplex` / `mcfftDoubleComplex` 数组。在 C2R 模式下，只需要包含非冗余复杂元素的输入数组 $(x_1, x_2, \ldots, x_{\lfloor \frac{N}{2} \rfloor + 1})$。输出数组 $(X_1, X_2, \ldots, X_N)$ 由 `mcfftReal`

/ mcfftDoubleReal 元素组成。最后，R2C 需要一个实数的输入数组 $(X_1, X_2, \ldots, X_N)$ 并返回一个非冗余复数元素的数组 $(x_1, x_2, \ldots, x_{\lfloor \frac{N}{2} \rfloor + 1})$。在实数到复数和复数到实数的变换中，输入数据的大小和输出数据的大小是不同的。对于异地变换，将创建一个大小适当的单独数组。对于原地变换，用户应该使用 padded 数据布局。这种布局是 FFTW 兼容的。

在 padded 布局中，输出信号开始于与输入数据相同的内存地址。因此，必须对实数到复数的输入数据和复数到实数的输出数据进行填充。

一维变换的输入/输出数据的预期大小总结如下表：

FFT 类型	输入数据大小	输出数据大小
C2C	x mcfftComplex	x mcfftComplex
C2R	$\lfloor \frac{x}{2} \rfloor + 1$ mcfftComplex	x mcfftReal
R2C*	x mcfftReal	$\lfloor \frac{x}{2} \rfloor + 1$ mcfftComplex

实数到复数的变换是一个正向变换。对于一个需要 FFTW 兼容输出的原地实数到复数变换，输入大小必须填充为 $(\lfloor \frac{N}{2} \rfloor + 1)$ 复数。对于异地变换，输入和输出大小分别匹配逻辑变换大小 N 和非冗余大小 $\lfloor \frac{N}{2} \rfloor + 1$。复数到实数变换是逆向变换。对于选择了 FFTW 兼容输出 (默认填充模式) 的原地复数到实数 FFTs 变换，假定输入大小为 $\lfloor \frac{N}{2} \rfloor + 1$ mcfftComplex 元素。注意，当选择非单元输入和输出步幅时，原地复数到实数 FFTs 变换可能会 **重写** 任意虚输入点值。异地复数到实数 FFT 变换将总是 **重写** 输入缓冲区。对于异地变换，输入和输出大小分别匹配逻辑变换非冗余大小 $\lfloor \frac{N}{2} \rfloor + 1$ 和大小 N。

2.3 多维变换

多维 DFT 将 d-维数组 $x_{\mathbf{n}}$ 映射到其频域数组，其中 $\mathbf{n} = (n_1, n_2, \ldots, n_d)$，由以下公式给出：

$$X_{\mathbf{k}} = \sum_{n=0}^{N-1} x_{\mathbf{n}} e^{-2\pi i \frac{\mathbf{kn}}{\mathbf{N}}}$$

其中 $\frac{\mathbf{n}}{\mathbf{N}} = (\frac{n_1}{N_1}, \frac{n_2}{N_2}, \ldots, \frac{n_d}{N_d})$，求和表示嵌套求和的集合

$$\sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} \cdots \sum_{n_d=0}^{N_d-1}$$

mcFFT 支持一维，二维和三维变换，它们都可以通过相同的 mcfftExec* 函数调用。

与一维情况类似，实数值输入数据的频域表示满足厄米特矩阵对称，定义为：$x_{(n_1,n_2,\ldots,n_d)} = x^*_{(N_1-n_1,N_2-n_2,\ldots,N_d-n_d)}$。C2R 和 R2C 算法利用该特性，只对信号数组的一半元素进行操作，即在：$x_{\mathbf{n}}$ 因为 $\mathbf{n} \in \{1, \ldots, N_1\} \times \ldots \times \{1, \ldots, N_{d-1}\} \times \{1, \ldots, \lfloor \frac{N_d}{2} \rfloor + 1\}$。在 Data Layout 中描述的数据对齐的一般规则适用于高维变换。下表总结了多维 DFTs 的输入和输出数据大小：

维度	FFT 类型	输入数据大小	输出数据大小
1D	C2C	$\mathbf{N}_1$ mcfftComplex	$\mathbf{N}_1$ mcfftComplex
1D	C2R	$\lfloor \frac{\mathbf{N}_1}{2} \rfloor + 1$ mcfftComplex	$\mathbf{N}_1$ mcfftReal
1D	R2C	$\mathbf{N}_1$ mcfftReal	$\lfloor \frac{\mathbf{N}_1}{2} \rfloor + 1$ mcfftComplex
2D	C2C	$\mathbf{N}_1\mathbf{N}_2$ mcfftComplex	$\mathbf{N}_1\mathbf{N}_2$ mcfftComplex
2D	C2R	$\mathbf{N}_1(\lfloor \frac{\mathbf{N}_2}{2} \rfloor + 1)$ mcfftComplex	$\mathbf{N}_1\mathbf{N}_2$ mcfftReal
2D	R2C	$\mathbf{N}_1\mathbf{N}_2$ mcfftReal	$\mathbf{N}_1(\lfloor \frac{\mathbf{N}_2}{2} \rfloor + 1)$ mcfftCompplex
3D	C2C	$\mathbf{N}_1\mathbf{N}_2\mathbf{N}_3$ mcfftComplex	$\mathbf{N}_1\mathbf{N}_2\mathbf{N}_3$ mcfftComplex
3D	C2R	$\mathbf{N}_1\mathbf{N}_2(\lfloor \frac{\mathbf{N}_3}{2} \rfloor + 1)$ mcfftComplex	$\mathbf{N}_1\mathbf{N}_2\mathbf{N}_3$ mcfftReal
3D	R2C	$\mathbf{N}_1\mathbf{N}_2\mathbf{N}_3$ mcfftReal	$\mathbf{N}_1\mathbf{N}_2(\lfloor \frac{\mathbf{N}_3}{2} \rfloor + 1)$ mcfftComplex

例如，对于一个异地变换的实数到复数变换输出的三维数组的静态声明如下所示：

```
mcfftComplex odata[N1][N2][N3/2+1];
```

3 mcFFT API 参考

本章通过描述其输入/输出参数、数据类型和错误代码来指定 mcFFT 库函数的行为。mcFFT 库在第一次调用 API 函数时初始化，并在销毁所有用户创建的 FFT 计划时自动关闭。

3.1 返回值 mcfftResult

除了 MCFFT_SUCCESS 之外，所有 mcFFT 库返回值都表明当前 API 调用失败，用户应该重新配置以纠正问题。可能的返回值定义如下：

```
typedef enum mcfftResult_t {
      MCFFT_SUCCESS         = 0,  //  mcFFT 操作成功
      MCFFT_INVALID_PLAN    = 1,  //  mcFFT 传递了一个无效的计划句柄
      MCFFT_ALLOC_FAILED    = 2,  //  mcFFT 分配 GPU 或 CPU 内存失败
      MCFFT_INVALID_TYPE    = 3,  //  不再使用
      MCFFT_INVALID_VALUE   = 4,  //  用户指定了无效的指针或参数
      MCFFT_INTERNAL_ERROR  = 5,  //  驱动程序或内部 mcFFT 库错误
      MCFFT_EXEC_FAILED     = 6,  //  在 GPU 上执行 FFT 失败
      MCFFT_SETUP_FAILED    = 7,  //  mcFFT 库初始化失败
      MCFFT_INVALID_SIZE    = 8,  //  用户指定了无效的变换大小
      MCFFT_UNALIGNED_DATA  = 9,  //  不再使用
      MCFFT_INCOMPLETE_PARAMETER_LIST = 10, //  调用中缺少参数
      MCFFT_INVALID_DEVICE  = 11, //  计划执行与计划创建在不同的 GPU 上
      MCFFT_PARSE_ERROR     = 12, //  内部计划数据库错误
      MCFFT_NO_WORKSPACE    = 13  //  在计划执行之前没有提供工作空间
      MCFFT_NOT_IMPLEMENTED = 14, // 函数未实现给定参数的功能
      MCFFT_LICENSE_ERROR   = 15, // 用于以前的版本
      MCFFT_NOT_SUPPORTED   = 16  // 不支持给定参数的操作
} mcfftResult;
```

鼓励用户检查 mcFFT 函数的返回值以查找错误。

3.2 mcFFT 基本计划

3.2.1 Function mcfftPlan1d()

```
mcfftResult
      mcfftPlan1d(mcfftHandle *plan, int nx, mcfftType type, int batch);
```

为指定的信号大小和数据类型创建一个一维 FFT 计划配置。batch 输入参数告诉 mcFFT 要配置多少个一维变换。

该调用对给定的句柄只能使用一次。如果计划已被锁定，即句柄之前被不同的 mcfftPlan 或 mcfftMakePlan 调用使用过，此调用将失败并返回 MCFFT_INVALID_PLAN。

输入

plan	指向一个 mcfftHandle 对象的指针
nx	变换大小 (例如，256 点 FFT 的大小为 256)
type	转换数据类型 (例如，MCFFT_C2C 用于将单精度复数转化为复数)
batch	大小为 nx 的变换次数。请考虑使用 mcfftPlanMany 进行多次变换。

输出

plan	包含一个 mcFFT 一维计划句柄值

返回值

MCFFT_SUCCESS	mcFFT 成功创建 FFT 计划。
MCFFT_INVALID_PLAN	plan 参数不是有效句柄。当计划被锁定，句柄无效。
MCFFT_ALLOC_FAILED	为计划分配 GPU 资源失败。
MCFFT_INVALID_VALUE	向 API 传递了一个或多个无效参数。
MCFFT_INTERNAL_ERROR	检测到内部驱动程序错误。
MCFFT_SETUP_FAILED	mcFFT 库初始化失败。
MCFFT_INVALID_SIZE	参数 nx 或 batch 为无效参数。

3.2.2 Function mcfftPlan2d()

```
mcfftResult
      mcfftPlan2d(mcfftHandle *plan, int nx, int ny, mcfftType type);
```

此调用根据指定的信号大小和数据类型创建二维 FFT 计划配置。

该调用对给定的句柄只能使用一次。如果计划已被锁定，即句柄之前被不同的 mcfftPlan 或 mcfftMakePlan 调用使用过，此调用将失败并返回 MCFFT_INVALID_PLAN。

输入

plan	指向一个 mcfftHandle 对象的指针
nx	在 x 维的变换大小。这是变换中变化最慢的维度 (跨距存储)。
ny	在 y 维的变换大小。这是变换中变化最快的维度 (连续存储)。
type	转换数据类型 (例如，MCFFT_C2R 用于表示单精度复数到实数的转换)

输出

plan	包含一个 mcFFT 二维计划句柄值

返回值

MCFFT_SUCCESS	mcFFT 成功创建 FFT 计划。
MCFFT_INVALID_PLAN	plan 参数不是有效句柄。当计划被锁定，句柄无效。
MCFFT_ALLOC_FAILED	为计划分配 GPU 资源失败。
MCFFT_INVALID_VALUE	向 API 传递了一个或多个无效参数。
MCFFT_INTERNAL_ERROR	检测到内部驱动程序错误。
MCFFT_SETUP_FAILED	mcFFT 库初始化失败。
MCFFT_INVALID_SIZE	参数 nx 或 ny 中至少存在一个大小不受支持的无效参数。

3.2.3 Function mcfftPlan3d()

```
mcfftResult
      mcfftPlan3d(mcfftHandle *plan, int nx, int ny, int nz, mcfftType
↪type);
```

根据指定的信号大小和数据类型创建三维 FFT 计划配置。除了第三个大小参数 nz 不同以外，这个函数与 mcfftPlan2d() 相同。

该调用对给定的句柄只能使用一次。如果计划已被锁定，即句柄之前被不同的 mcfftPlan 或 mcfftMakePlan 调用使用过，此调用将失败并返回 MCFFT_INVALID_PLAN。

输入

plan	指向一个 mcfftHandle 对象的指针。
nx	在 x 维的变换大小。这是变换中变化最慢的维度 (跨距存储)。
ny	在 y 维的变换大小。
nz	在 z 维的变换大小。这是变换中变化最快的维度 (连续存储)。
type	转换数据类型 (例如，MCFFT_R2C 用于表示单精度实数到复数的变换)

输出

plan	包含一个 mcFFT 三维计划句柄值

返回值

MCFFT_SUCCESS	mcFFT 成功创建 FFT 计划。
MCFFT_INVALID_PLAN	plan 参数不是有效句柄。当计划被锁定，句柄无效。
MCFFT_ALLOC_FAILED	为计划分配 GPU 资源失败。
MCFFT_INVALID_VALUE	向 API 传递了一个或多个无效参数。
MCFFT_INTERNAL_ERROR	检测到内部驱动程序错误。
MCFFT_SETUP_FAILED	mcFFT 库初始化失败。
MCFFT_INVALID_SIZE	参数 nx、ny 或 nz 中至少存在一个大小不受支持的无效参数。

3.2.4 Function mcfftPlanMany()

```
mcfftResult
      mcfftPlanMany(mcfftHandle *plan, int rank, int *n, int *inembed,
         int istride, int idist, int *onembed, int ostride,
         int odist, mcfftType type, int batch);
```

创建维度 rank 的一个 FFT 计划配置，其大小由数组 n 指定。输入参数 batch 告诉 mcFFT 需要配置多少个变换。通过这个函数，可以创建一维、二维或三维的批处理计划。

mcfftPlanMany() API 通过高级数据布局参数支持更复杂的输入和输出数据布局：inembed，istride，idist，onembed，ostride，和 odist。

如果 inembed 和 onembed 被设置为 NULL，其他所有步幅信息将被忽略，并使用默认步幅。默认情况下假定连续的数据数组。

所有数组都假定在 CPU 内存中。

请注意，当 inembed 和 onembed 为 NULL 时，mcfftPlanMany 函数的行为与 FFTW 库 fftw_plan_many_dft 中对应的函数不同。

该调用对给定的句柄只能使用一次。如果计划已被锁定，即句柄之前被不同的 mcfftPlan 或 mcfftMakePlan 调用使用过，此调用将失败并返回 MCFFT_INVALID_PLAN。

输入

plan	指向一个 mcfftHandle 对象的指针。
rank	变换的维度 (1、2 或 3)。
n	大小为 rank 的数组描述每个维度的大小，n[0] 表示的大小。
inembed	大小为 rank 的指针表示输入数据在内存中的存储维度。如果设置为 NULL，则忽略其他所有高级数据布局参数。
istride	表示两个连续输入元素在最低有效维度 (即最内层) 之间的距离。
idist	表示一批输入数据中两个连续信号的第一个元素之间的距离
onembed	大小为 rank 的指针表示输出数据在内存中的存储规模。如果设置为 NULL，则忽略其他所有高级数据布局参数。
ostride	表示输出数组中两个连续输出的元素在最低有效维度 (即最内层) 之间的距离。
odist	表示批处理变换输入数据中两个连续信号的第一个元素之间的距离。
type	转换数据类型 (例如，MCFFT_R2C 用于表示单精度实数到复数的变换)。
batch	此变换的批处理大小。

输出

plan	包含一个 mcFFT 计划句柄。

返回值

MCFFT_SUCCESS	mcFFT 成功创建 FFT 计划。
MCFFT_INVALID_PLAN	plan 参数不是有效句柄。当计划被锁定，句柄无效。
MCFFT_ALLOC_FAILED	为计划分配 GPU 资源失败。
MCFFT_INVALID_VALUE	向 API 传递了一个或多个无效参数
MCFFT_INTERNAL_ERROR	检测到内部驱动程序错误。
MCFFT_SETUP_FAILED	mcFFT 库初始化失败。
MCFFT_INVALID_SIZE	至少存在一个大小不受支持的无效参数。

3.3 mcFFT 扩展计划

该 API 将句柄创建与计划生成分离开来。这使得在计划实际生成之前更改计划的设置成为可能，从而可能会改变计划生成阶段的结果。

3.3.1 Function mcfftCreate()

```
mcfftResult
      mcfftCreate(mcfftHandle *plan);
```

仅创建不透明句柄，并在主机上分配小型数据结构。由 mcfftMakePlan*() 调用来实际执行计划。

输入

plan	指向 mcfftHandle 对象的指针。

输出

plan	包含一个 mcFFT 计划句柄值。

返回值

MCFFT_SUCCESS	mcFFT 成功创建 FFT 计划。
MCFFT_ALLOC_FAILED	为计划分配资源失败。
MCFFT_INVALID_VALUE	向 API 传递了一个或多个无效参数
MCFFT_INTERNAL_ERROR	检测到内部驱动程序错误。
MCFFT_SETUP_FAILED	mcFFT 库初始化失败。

3.3.2 Function mcfftMakePlan1d()

```
mcfftResult
      mcfftMakePlan1d(mcfftHandle plan, int nx, mcfftType type, int batch,
         size_t *workSize);
```

在调用 mcfftCreate() 之后，为指定的信号大小和数据类型创建一个一维 FFT 计划配置。batch 输入参数告诉 mcFFT 要配置多少个一维变换。

该调用对给定的句柄只能使用一次。如果计划已被锁定，即句柄之前被不同的 mcfftPlan 或 mcfftMakePlan 调用使用过，此调用将失败并返回 MCFFT_INVALID_PLAN。

输入

plan	由 mcfftCreate 返回的 mcfftHandle。
nx	变换大小 (例如，256 点 FFT 的大小为 256)。
type	转换数据类型 (例如，MCFFT_C2C 用于表示单精度复数到复数的转换。)
batch	大小为 nx 的变换次数。多次转换的情况下，请考虑使用 mcfftMakePlanMany 进行。
*workSize	指向工作区大小的指针，以字节为单位。

输出

*workSize	指向工作区大小的指针。

返回值

MCFFT_SUCCESS	mcFFT 成功创建 FFT 计划。
MCFFT_INVALID_PLAN	plan 参数不是有效句柄。当计划被锁定或不满足多 GPU 限制，句柄无效。
MCFFT_ALLOC_FAILED	为计划分配 GPU 内存失败。
MCFFT_INVALID_VALUE	向 API 传递一个或多个无效参数。
MCFFT_INTERNAL_ERROR	检测到内部驱动程序错误。
MCFFT_SETUP_FAILED	mcFFT 库初始化失败。
MCFFT_INVALID_SIZE	参数 nx 或 batch 为无效参数。

3.3.3 Function mcfftMakePlan2d()

```
mcfftResult
      mcfftMakePlan2d(mcfftHandle plan, int nx, int ny, mcfftType type,
         size_t *workSize);
```

在调用 mcfftCreate() 之后，根据指定的信号大小和数据类型创建一个二维 FFT 计划配置。

该调用对给定的句柄只能使用一次。如果计划已被锁定，即句柄之前被不同的 mcfftPlan 或 mcfftMakePlan 调用使用过，此调用将失败并返回 MCFFT_INVALID_PLAN。

输入

plan	由 mcfftCreate 返回的 mcfftHandle。
nx	在 x 维度的变换大小。这是变换中最慢变化的维度。(跨距存储)。
ny	在 y 维度的变换大小。这是变换中最快变化的维度。(连续存储)。
type	变换的数据类型 (例如，MCFFT_C2R 用于表示单精度复数到实数的变换)。
*workSize	指向工作区大小的指针，以字节为单位。

输出

*workSize	指向工作区大小的指针。

返回值

MCFFT_SUCCESS	mcFFT 成功创建了 FFT 计划。
MCFFT_INVALID_PLAN	plan 参数不是有效句柄。
MCFFT_ALLOC_FAILED	为计划分配 GPU 资源失败。
MCFFT_INVALID_VALUE	向 API 传递了一个或多个无效参数。
MCFFT_INTERNAL_ERROR	检测到内部驱动程序错误。
MCFFT_SETUP_FAILED	mcFFT 库初始化失败。
MCFFT_INVALID_SIZE	参数 nx 或 ny 中至少存在一个大小不受支持的无效参数。

3.3.4 Function mcfftMakePlan3d()

```
mcfftResult
      mcfftMakePlan3d(mcfftHandle plan, int nx, int ny, int nz, mcfftType
↪type,
         size_t *workSize);
```

调用 mcfftCreate() 后，根据指定的信号大小和数据类型进行三维 FFT 规划配置。除了第三个大小参数 nz 以外，这个函数与 mcfftPlan2d() 相同。

该调用对给定的句柄只能使用一次。如果计划已被锁定，即句柄之前被不同的 mcfftPlan 或 mcfftMakePlan 调用使用过，此调用将失败并返回 MCFFT_INVALID_PLAN。

输入

plan	由 mcfftCreate 返回的 mcfftHandle。
nx	在 x 维度的变换大小。这是变换中变化最慢的维度。(跨距存储)。
ny	在 y 维度上的变换大小。
nz	在 z 维度上的变换大小。这是变换的变化最快维度。(连续存储)。
type	变换的数据类型 (例如，MCFFT_R2C 用于表示单精度复数到实数的变换)。
*workSize	指向工作区大小的指针，以字节为单位。

输出

*workSize	指向工作区大小的指针。

返回值

MCFFT_SUCCESS	mcFFT 成功创建了 FFT 计划。
MCFFT_INVALID_PLAN	plan 参数不是有效句柄。
MCFFT_ALLOC_FAILED	为计划分配 GPU 资源失败。
MCFFT_INVALID_VALUE	向 API 传递了一个或多个无效参数。
MCFFT_INTERNAL_ERROR	检测到内部驱动程序错误。
MCFFT_SETUP_FAILED	mcFFT 库初始化失败。
MCFFT_INVALID_SIZE	参数 nx、ny 或 nz 中至少存在一个大小不受支持的无效参数。

3.3.5 Function mcfftMakePlanMany()

```
mcfftResult
      mcfftMakePlanMany(mcfftHandle plan, int rank, int *n, int *inembed,
         int istride, int idist, int *onembed, int ostride,
         int odist, mcfftType type, int batch, size_t *workSize);
```

在调用 mcfftCreate() 之后，会创建一个维度为 rank 的 FFT 规划配置，其大小由数组 n 指定。输入参数 batch 告诉 mcFFT 需要配置多少个变换。通过这个函数，可以创建一维、二维或三维的批处理计划。

mcfftPlanMany() API 通过高级数据布局参数支持更复杂的输入和输出数据布局：inembed、istride、idist、onembed、ostride 和 odist。

如果 inembed 和 onembed 设置为 NULL，则其他步幅信息将被忽略，并且使用默认步幅。默认情况下假设数据数组是连续的。

该调用对给定的句柄只能使用一次。如果计划已被锁定，即句柄之前被不同的 mcfftPlan 或 mcfftMakePlan 调用使用过，此调用将失败并返回 MCFFT_INVALID_PLAN。

所有数组被假定在 CPU 内存中

输入

plan	由 mcfftCreate 返回的 mcfftHandle。
rank	变换的维度 (1，2，或 3)。
n	大小为 rank 的数组，描述每个维度的大小。n[0] 表示最外层维度的大小，n[rank-1] 表示最内层 (连续) 维度的大小。
inembed	大小为 rank 的指针，用于指示内存中输入数据的存储维度。inembed[0] 是最外层维度的存储维度。如果设置为 NULL，则忽略所有其他高级数据布局参数。
istride	表示两个连续输入元素在最低有效维度 (即最内层) 之间的距离
idist	表示一批输入数据中两个连续信号的第一个元素之间的距离
onembed	大小为 rank 的指针，用于指示内存中输入数据的存储维度。inembed[0] 是最外层维度的存储维度。如果设置为 NULL，则忽略所有其他高级数据布局参数。
ostride	表示两个连续输出元素在最低有效维度 (即最内层) 之间的距离
odist	表示一批输出数据中两个连续信号的第一个元素之间的距离
type	变换的数据类型 (例如，MCFFT_R2C 用于表示单精度实数到复数的变换)
batch	此变换的单批数据的大小。
*workSize	指向工作区大小的指针，以字节为单位。

输出

*workSize	指向工作区大小的指针。

返回值

MCFFT_SUCCESS	mcFFT 成功创建了 FFT 计划。
MCFFT_INVALID_PLAN	plan 参数不是有效句柄。当计划被锁定或不满足多 GPU 限制时，句柄无效。
MCFFT_ALLOC_FAILED	为计划分配 GPU 资源失败。
MCFFT_INVALID_VALUE	向 API 传递了一个或多个无效参数。
MCFFT_INTERNAL_ERROR	检测到内部驱动程序错误。
MCFFT_SETUP_FAILED	mcFFT 库初始化失败。
MCFFT_INVALID_SIZE	至少存在一个大小不受支持的无效参数。

3.3.6 Function mcfftMakePlanMany64()

```
mcfftResult
      mcfftMakePlanMany64(mcfftHandle plan, int rank,
         long long int *n,
         long long int *inembed, long long int istride, long long int
↪idist,
         long long int *onembed, long long int ostride, long long int
↪odist,
         mcfftType type,
         long long int batch, size_t *workSize);
```

在调用 mcfftCreate() 之后，会创建一个维度为 rank 的 FFT 规划配置，其大小由数组 n 指定。输入参数 batch 告诉 mcFFT 需要配置多少个变换。通过这个函数，可以创建一维、二维或三维的批处理计划。

这个 API 与 mcfftMakePlanMany 完全相同，只是指定尺寸和步幅的参数是 64 位整数。这个 API 使得大规模变换成为可能。mcFFT 包括使用 32 位索引和使用 64 位索引的内核。mcFFT 计划在可能的情况下选择使用 32 位内核，以避免由于 64 位算术而引起的任何开销。

此接口支持所有大小和类型的变换，只有两个例外。对于大小超过 4G 元素的变换，数组 n 中指定的维度必须是小于等于 17 的可分解质数。对于大小超过 4G 元素的实数到复数和复数到实数变换，最快变化的维度必须是偶数。

mcfftPlanMany64() API 通过高级数据布局参数支持更复杂的输入和输出数据布局：inembed、istride、idist、onembed、ostride 和 odist。

如果 inembed 和 onembed 被设置为 NULL，则其他步幅信息将被忽略，并且使用默认步幅。默认情况下假设数据数组是连续的。

该调用对给定的句柄只能使用一次。如果计划已被锁定，即句柄之前被不同的 mcfftPlan 或 mcfftMakePlan 调用使用过，此调用将失败并返回 MCFFT_INVALID_PLAN。

所有数组被假定在 CPU 内存中。

输入

plan	由 mcfftCreate 返回的 mcfftHandle。
rank	变换的维度 (1，2，或 3)。
n	大小为 rank 的数组，描述每个维度的大小。
inembed	大小为 rank 的指针，用于指示内存中输入数据的存储维度。如果设置为 NULL，则忽略所有其他高级数据布局参数。
istride	表示两个连续输入元素在最低有效维度 (即最内层) 之间的距离。
idist	表示一批输入数据中两个连续信号的第一个元素之间的距离
onembed	大小为 rank 的指针，表示输出数据在内存中的存储维度。如果设置为 NULL，则忽略所有其他高级数据布局参数。
ostride	表示两个连续输出元素在最低有效维度 (即最内层) 之间的距离
odist	表示一批输出数据中两个连续信号的第一个元素之间的距离
type	变换的数据类型 (例如，MCFFT_R2C 用于表示单精度实数到复数的变换)
batch	此变换的单批数据的大小
*workSize	指向工作区大小的指针，以字节为单位

输出

*workSize	指向工作区大小的指针。

返回值

MCFFT_SUCCESS	mcFFT 成功创建了 FFT 计划。
MCFFT_INVALID_PLAN	plan 参数不是有效句柄。当计划被锁定或不满足多 GPU 限制时，句柄无效。
MCFFT_ALLOC_FAILED	为计划分配 GPU 资源失败。
MCFFT_INVALID_VALUE	向 API 传递了一个或多个无效参数。
MCFFT_INTERNAL_ERROR	检测到内部驱动程序错误。
MCFFT_SETUP_FAILED	mcFFT 库初始化失败。
MCFFT_INVALID_SIZE	至少存在一个大小不受支持的无效参数。

3.4 mcFFT 估算工作区大小

在执行计划过程中，mcFFT 需要一个工作区来临时存储中间结果。mcfftEstimate*() 函数调用根据指定的参数和默认计划设置返回所需工作区大小的估计值。某些问题需要比其他问题更多的存储空间。就临时存储而言，二次幂尤其高效。然而，大的素数使用不同的算法，可能需要多达类似大小的二次幂的 8 倍存储空间。这些函数返回的 workSize 值可能仍然小于实际需要的值，尤其是对于不是 2、3、5 和 7 次幂倍数的 n 值。更精确的值可以通过 mcfftGetSize*() 函数获得，但这些值可能仍然相对保守。

3.4.1 Function mcfftEstimate1d()

```
mcfftResult
      mcfftEstimate1d(int nx, mcfftType type, int batch, size_t *workSize);
```

在计划执行过程中，mcFFT 需要一个工作区来临时存储中间结果。该函数调用根据指定的参数和假设的默认计划设置返回所需工作区大小的估计值。

输入

nx	变换的大小 (例如，256 表示 256 点 FFT)。
type	变换的数据类型 (例如 MCFFT_C2C 表示单精度复数到复数的变换)。
batch	大小为 nx 的变换的数量。对于多个变换，请考虑使用 mcfftEstimateMany 函数。
*workSize	指向工作区大小的指针，以字节为单位。

输出

*workSize	指向工作区大小的指针。

返回值

MCFFT_SUCCESS	mcFFT 成功返回了工作区大小。
MCFFT_ALLOC_FAILED	为计划分配 GPU 资源失败。
MCFFT_INVALID_VALUE	向 API 传递了一个或多个无效参数。
MCFFT_INTERNAL_ERROR	检测到内部驱动程序错误。
MCFFT_SETUP_FAILED	mcFFT 库初始化失败。
MCFFT_INVALID_SIZE	nx 参数是无效参数。

3.4.2 Function mcfftEstimate2d()

```
mcfftResult
      mcfftEstimate2d(int nx, int ny, mcfftType type, size_t *workSize);
```

在计划执行过程中，mcFFT 需要一个工作区来临时存储中间结果。该函数调用根据指定的参数和假设的默认计划设置返回所需工作区大小的估计值。

输入

nx	在 x 维度上的变换大小 (行数)。
ny	在 y 维度上的变换大小 (列数)。
type	变换的数据类型 (例如，MCFFT_C2R 表示单精度复数到实数的变换)。
*workSize	指向工作区大小的指针，以字节为单位。

输出

*workSize	指向工作区大小的指针。

返回值

MCFFT_SUCCESS	mcFFT 成功返回了工作区的大小。
MCFFT_INVALID_PLAN	plan 参数不是有效的句柄。
MCFFT_ALLOC_FAILED	为计划分配 GPU 资源失败。
MCFFT_INVALID_VALUE	向 API 传递了一个或多个无效参数。
MCFFT_INTERNAL_ERROR	检测到内部驱动程序错误。
MCFFT_SETUP_FAILED	mcFFT 库初始化失败。
MCFFT_INVALID_SIZE	参数 nx 或 ny 中至少存在一个大小不受支持的无效参数。

3.4.3 Function mcfftEstimate3d()

```
mcfftResult
      mcfftEstimate3d(int nx, int ny, int nz, mcfftType type, size_t
↪*workSize);
```

在计划执行过程中，mcFFT 需要一个工作区来临时存储中间结果。该函数调用根据指定的参数和假设的默认计划设置返回所需工作区大小的估计值。

输入

nx	在 x 维度上的变换大小。
ny	在 y 维度上的变换大小。
nz	在 z 维度上的变换大小。
type	变换的数据类型 (例如，MCFFT_R2C 表示单精度实数到复数的变换)。
*workSize	指向工作区大小的指针，以字节为单位。

输出

*workSize	指向工作区大小的指针。

Return Values

MCFFT_SUCCESS	mcFFT 成功返回了工作区大小。
MCFFT_INVALID_PLAN	plan 参数不是有效的句柄。
MCFFT_ALLOC_FAILED	为计划分配 GPU 资源失败。
MCFFT_INVALID_VALUE	向 API 传递了一个或多个无效参数。
MCFFT_INTERNAL_ERROR	检测到内部驱动程序错误。
MCFFT_SETUP_FAILED	mcFFT 库初始化失败。
MCFFT_INVALID_SIZE	参数 nx、ny 或 nz 中至少存在一个大小不受支持的无效参数。

3.4.4 Function mcfftEstimateMany()

```
mcfftResult
    mcfftEstimateMany(int rank, int *n, int *inembed,
        int istride, int idist, int *onembed, int ostride,
        int odist, mcfftType type, int batch, size_t *workSize);
```

在计划执行过程中，mcFFT 需要一个工作区来临时存储中间结果。该函数调用根据指定的参数和假设的默认计划设置返回所需工作区大小的估计值。

mcfftPlanMany() API 通过高级数据布局参数支持更复杂的输入和输出数据布局：inembed、istride、idist、onembed、ostride 和 odist。

假定所有数组在 GPU 内存中。

输入

rank	变换的维度 (1，2，或 3)。
n	大小为 rank 的数组, 描述每个维度的大小。
inembed	大小为 rank 的指针，用于指示内存中输入数据的存储维度。如果设置为 NULL，则忽略所有其他高级数据布局参数。
istride	表示两个连续输入元素在最低有效维度 (即最内层) 之间的距离
idist	表示一批输入数据中两个连续信号的第一个元素之间的距离
onembed	大小为 rank 的指针，表示输出数据在内存中的存储维度。如果设置为 NULL，则忽略所有其他高级数据布局参数。
ostride	表示两个连续输出元素在最低有效维度 (即最内层) 之间的距离
odist	表示批处理变换输出数据中两个连续信号的第一个元素之间的距离。
type	变换的数据类型 (例如，MCFFT_R2C 用于表示单精度实数到复数的变换)
batch	此变换的批处理大小。
*workSize	指向工作区大小的指针，以字节为单位。

输出

*workSize	指向工作区大小的指针。

返回值

MCFFT_SUCCESS	mcFFT 成功创建了 FFT 计划。
MCFFT_ALLOC_FAILED	为计划分配 GPU 资源失败。
MCFFT_INVALID_VALUE	向 API 传递了一个或多个无效参数。
MCFFT_INTERNAL_ERROR	检测到内部驱动程序错误。
MCFFT_SETUP_FAILED	mcFFT 库初始化失败。
MCFFT_INVALID_SIZE	至少存在一个大小不受支持的无效参数。

3.5 优化后 mcFFT 估算工作区大小

mcfftGetSize*() 函数比 mcfftEstimate*() 函数提供了更准确的计划所需工作区大小的估计，因为在于考虑了可能已进行的任何计划设置。正如在 mcFFT 估算工作区大小部分中所讨论的那样，返回的 workSize 值可能过于保守，尤其是当 n 不是 2、3、5 和 7 的幂的倍数的情况。

3.5.1 Function mcfftGetSize1d()

```
mcfftResult
      mcfftGetSize1d(mcfftHandle plan, int nx, mcfftType type, int batch,
         size_t *workSize);
```

与 mcfftEstimate1d() 相比，在给定参数并考虑了可能已经进行的任何计划设置的情况下，mcfftGetSize1d() 调用给出的计划所需工作区大小更为准确。

输入

plan	由 mcfftCreate 返回的 mcfftHandle。
nx	变换的大小 (例如，256 表示 256 点 FFT)。
type	变换的数据类型 (例如，单精度复数到复数的 MCFFT_C2C)。
batch	大小为 nx 的变换数量。请考虑使用 mcfftGetSizeMany 进行多个变换。
*workSize	指向工作区大小的指针，以字节为单位。

输出

*workSize	指向工作区大小的指针。

返回值

MCFFT_SUCCESS	mcFFT 成功返回了工作区大小。
MCFFT_INVALID_PLAN	plan 参数不是有效句柄。
MCFFT_ALLOC_FAILED	为计划分配 GPU 资源失败。
MCFFT_INVALID_VALUE	向 API 传递了一个或多个无效参数。
MCFFT_INTERNAL_ERROR	检测到内部驱动程序错误。
MCFFT_SETUP_FAILED	mcFFT 库初始化失败。
MCFFT_INVALID_SIZE	参数 nx 为无效参数。

3.5.2 Function mcfftGetSize2d()

```
mcfftResult
      mcfftGetSize2d(mcfftHandle plan, int nx, int ny, mcfftType type,
         size_t *workSize);
```

在指定参数并考虑到可能的任何计划设置的情况下，调用 mcfftGetSize2d() 会比 mcfftEstimate2d() 更准确地估算计划所需的工作区大小。

输入

plan	mcfftCreate 返回的 mcfftHandle。
nx	X 维度上的变换大小 (行数)。
ny	Y 维度上的变换大小 (列数)。
type	转换数据类型 (例如 MCFFT_C2R 表示单精度复数转换为实数)。
*workSize	指向工作区大小的指针，以字节为单位。

输出

*workSize	指向工作区大小的指针。

返回值

MCFFT_SUCCESS	mcFFT 成功返回了工作区的大小。
MCFFT_INVALID_PLAN	plan 参数不是有效句柄。
MCFFT_ALLOC_FAILED	为计划分配 GPU 资源失败。
MCFFT_INVALID_VALUE	向 API 传递了一个或多个无效参数。
MCFFT_INTERNAL_ERROR	检测到一个内部驱动程序错误。
MCFFT_SETUP_FAILED	mcFFT 库初始化失败。
MCFFT_INVALID_SIZE	参数 nx 或 ny 中至少存在一个大小不受支持的无效参数。

3.5.3 Function mcfftGetSize3d()

```
mcfftResult
      mcfftGetSize3d(mcfftHandle plan, int nx, int ny, int nz, mcfftType
↪type,
         size_t *workSize);
```

在指定参数并考虑到任何可能的计划设置的情况下，调用 mcfftGetSize3d() 会比 mcfftEstimate3d() 更准确地估算计划所需的工作区大小。

输入

plan	由 mcfftCreate 返回的 mcfftHandle。
nx	在 x 维度上的变换大小。
ny	在 y 维度上的变换大小。
nz	在 z 维度上的变换大小。
type	转换数据类型 (例如 MCFFT_C2R 表示单精度复数转换为实数)。
*workSize	指向工作区大小的指针 (字节)。

输出

*workSize	指向工作区大小的指针。

返回值

MCFFT_SUCCESS	mcFFT 成功返回了工作区的大小。
MCFFT_INVALID_PLAN	plan 参数不是有效句柄。
MCFFT_ALLOC_FAILED	为计划分配 GPU 资源失败。
MCFFT_INVALID_VALUE	向 API 传递了一个或多个无效参数。
MCFFT_INTERNAL_ERROR	检测到一个内部驱动程序错误。
MCFFT_SETUP_FAILED	mcFFT 库初始化失败。
MCFFT_INVALID_SIZE	至少存在一个大小不受支持的无效参数。

3.5.4 Function mcfftGetSizeMany()

```
mcfftResult
      mcfftGetSizeMany(mcfftHandle plan, int rank, int *n, int *inembed,
        int istride, int idist, int *onembed, int ostride,
        int odist, mcfftType type, int batch, size_t *workSize);
```

在指定参数并考虑到任何可能的计划设置的情况下，该调用会比 mcfftEstimateSizeMany() 更准确地估算计划所需的工作区大小。

输入

plan	mcfftCreate 返回的 mcfftHandle。
rank	转换的维度（1、2 或 3)。
n	表示 rank 大小的数组，描述每个维度的大小。
inembed	指向 rank 大小的指针，表示输入数据在内存中的存储大小。如果设置为 NULL，则忽略所有其他高级数据布局参数。
istride	表示两个连续输入元素在最低有效维度 (即最内层) 上的距离。
idist	表示一批输入数据中两个连续信号彼此之间第一个元素相距的距离。
onembed	指向 rank 大小的指针，表示输出数据在内存中的存储大小。如果设置为 NULL，则忽略所有其他高级数据布局参数。
ostride	该参数表示输出数组中相邻输出元素在最低有效维度 (即最内层) 上的间距。
odist	表示一批输出数据中两个连续信号彼此之间第一个元素相距的距离。
type	变换数据类型 (例如，可以使用 MCFFT_R2C 来进行单精度实数到复数的变换)。
batch	这个变换的批量大小。
*workSize	指向工作区大小（以字节为单位）的指针。

输出

*workSize	指向工作区大小的指针。

返回值

MCFFT_SUCCESS	mcFFT 成功返回了工作区的大小。
MCFFT_INVALID_PLAN	plan 参数不是有效的句柄。
MCFFT_ALLOC_FAILED	为该计划分配 GPU 资源失败。
MCFFT_INVALID_VALUE	向 API 传递了一个或多个无效参数。
MCFFT_INTERNAL_ERROR	检测到一个内部驱动程序错误。
MCFFT_SETUP_FAILED	mcFFT 库初始化失败。
MCFFT_INVALID_SIZE	存在一个或多个无效参数。

3.5.5 Function mcfftGetSizeMany64()

```
mcfftResult
      mcfftGetSizeMany64(mcfftHandle plan, int rank,
         long long int *n,
         long long int *inembed, long long int istride, long long int
↪idist,
         long long int *onembed, long long int ostride, long long int
↪odist,
         mcfftType type,
         long long int batch, size_t *workSize);
```

在指定参数并考虑到任何可能计划设置的情况下，调用 mcfftGetSizeMany64() 会比 mcfftEstimateSizeMany() 更准确地估算计划所需的工作区大小。

这个 API 与 mcfftMakePlanMany 相同，只是指定大小和步幅的参数是 64 位整数。这个 API 使得大规模变换成为可能。mcFFT 可以使用 32 位索引或 64 位索引的内核。mcFFT 在可能的情况下会选择使用 32 位内核，以避免由于 64 位运算而产生额外的开销。

这个接口支持所有大小和类型的变换，但有两个例外。首先对于总大小超过 4G 个元素的变换，数组 n 中指定的维度必须能够分解为小于或等于 17 的质数。其次对于总大小超过 4G 个元素的实数到复数与复数到实数的变换，最快变化的维度必须是偶数。

输入

plan	由 mcfftCreate 返回的 mcfftHandle。
rank	转换的维度 (1、2 或 3)。
n	表示 rank 大小的数组，描述每个维度的大小。
inembed	指向 rank 大小的指针，表示输入数据在内存中的存储大小。如果设置为 NULL，所有其他高级数据布局参数都将被忽略。
istride	表示两个连续输入元素在最低有效维度 (即最内层) 上的距离。
idist	表示一批输入数据中两个连续信号的第一个元素之间的距离
onembed	指向大小为 rank 的指针，用于指示内存中输出数据的存储维度。如果设置为 NULL，则忽略所有其他高级数据布局参数。
ostride	该参数表示输出数组中相邻输出元素在最低有效维度 (即最内层) 上的间距。
odist	表示一批输出数据中两个连续信号的第一个元素之间的距离
type	变换数据类型 (例如，对于单精度实数到复数的变换，可以使用 MCFFT_R2C)。
batch	此变换的单批数据的大小。
*workSize	指向工作区大小的指针，以字节为单位。

输出

*workSize	指向工作区大小的指针。

返回值

MCFFT_SUCCESS	mcFFT 成功返回了工作区的大小。
MCFFT_INVALID_PLAN	plan 参数不是有效句柄。
MCFFT_ALLOC_FAILED	为计划分配 GPU 资源失败。
MCFFT_INVALID_VALUE	向 API 传递了一个或多个无效参数。
MCFFT_INTERNAL_ERROR	检测到一个内部驱动程序错误。
MCFFT_SETUP_FAILED	mcFFT 库初始化失败。
MCFFT_INVALID_SIZE	至少存在一个大小不受支持的无效参数。

3.6 Function mcfftGetSize()

```
mcfftResult
      mcfftGetSize(mcfftHandle plan, size_t *workSize);
```

此调用在使用原始 API 或扩展 API 生成计划后，会返回该计划所需的实际工作区大小。选择在其应用程序中管理工作区分配的调用者必须在计划生成后使用此调用，如果在计划生成后调用任意 mcfftSet*() ，可能会改变所需的工作区大小，也必须使用此调用。

输入

plan	由 mcfftCreate 返回的 mcfftHandle。
*workSize	指向工作区大小的指针，以字节为单位。

输出

*workSize	指向工作区大小的指针。

返回值

MCFFT_SUCCESS	mcFFT 成功返回了工作区的大小。
MCFFT_INVALID_PLAN	plan 参数不是有效句柄。
MCFFT_INTERNAL_ERROR	为计划分配 GPU 资源失败。

3.7 mcFFT 调用器分配工作区支持

3.7.1 Function mcfftSetAutoAllocation()

```
mcfftResult
      mcfftSetAutoAllocation(mcfftHandle plan, int autoAllocate);
```

mcfftSetAutoAllocation() 表示调用者打算为已生成的计划分配和管理工作区。mcFFT 默认在计划生成时分配工作区。如果在调用任意 mcfftMakePlan*() 之前调用了 mcfftSetAutoAllocation()，并将 autoAllocate 设置为 0（false），则 mcFFT 不会分配工作区。对于希望管理工作区域分配的调用者来说，这是首选顺序。

输入

plan	由 mcfftCreate 返回的 mcfftHandle。
autoAllocate	表示是否分配工作区。

返回值

MCFFT_SUCCESS	mcFFT 成功返回了工作区的大小。
MCFFT_INVALID_PLAN	plan 参数不是有效句柄。
MCFFT_INTERNAL_ERROR	检测到一个内部驱动程序错误。

3.7.2 Function mcfftSetWorkArea()

```
mcfftResult
      mcfftSetWorkArea(mcfftHandle plan, void *workArea);
```

mcfftSetWorkArea() 重写了与计划相关联的工作区指针。如果工作区是自动分配的，mcFFT 会释放自动分配的空间。mcfftExecute*() 调用假设工作区指针是有效的，并且它指向设备内存中的一个连续区域，不与任何其他工作区重叠。无法保证该情况以外的结果。

输入

plan	由 mcfftCreate 返回的 mcfftHandle。
workArea	指向工作区的指针。

返回值

MCFFT_SUCCESS	mcFFT 成功允许用户覆盖工作区指针。
MCFFT_INVALID_PLAN	plan 参数不是有效句柄。
MCFFT_INTERNAL_ERROR	检测到内部驱动程序错误。
MCFFT_SETUP_FAILED	mcFFT 库初始化失败。

3.8 Function mcfftDestroy()

```
mcfftResult
      mcfftDestroy(mcfftHandle plan);
```

释放与 mcFFT 计划相关的所有 GPU 资源，并销毁内部计划数据结构。一旦不再需要计划，mcfftDestroy() 就被调用以避免浪费 GPU 内存。

输入

plan	要销毁的计划中 mcfftHandle 对象。

返回值

MCFFT_SUCCESS	mcFFT 成功销毁了 FFT 计划。
MCFFT_INVALID_PLAN	plan 参数不是一个有效句柄

3.9 执行 mcFFT

3.9.1 Function mcfftExecC2C() 和 mcfftExecZ2Z()

```
mcfftResult
      mcfftExecC2C(mcfftHandle plan, mcfftComplex *idata,
         mcfftComplex *odata, int direction);
mcfftResult
      mcfftExecZ2Z(mcfftHandle plan, mcfftDoubleComplex *idata,
         mcfftDoubleComplex *odata, int direction);
```

`mcfftExecC2C()` （`mcfftExecZ2Z()`）函数根据 `direction` 参数指定的变换方向执行单精度 (双精度) 复数到复数的变换计划。mcFFT 使用由 `idata` 参数指向的 GPU 内存作为输入数据。该函数将傅里叶系数存储在 `odata` 数组中。如果 `idata` 和 `odata` 相同，这个方法将执行原地变换。

输入

`plan`	由 `mcfftCreate` 返回的 `mcfftHandle`。
`idata`	指向要进行变换的复数输入数据 (存储在 GPU 内存中) 的指针。
`odata`	指向复数输出数据 (存储在 GPU 内存中) 的指针
`direction`	变换路径：`MCFFT_FORWARD` 或 `MCFFT_INVERSE`。

输出

`odata`	包含复数傅里叶系数。

返回值

`MCFFT_SUCCESS`	mcFFT 成功执行了 FFT 计划。
`MCFFT_INVALID_PLAN`	`plan` 参数不是一个有效的句柄。
`MCFFT_INVALID_VALUE`	`idata`，`odata`，`direction` 中至少存在一个无效参数。
`MCFFT_INTERNAL_ERROR`	检测到内部驱动程序错误。
`MCFFT_EXEC_FAILED`	mcFFT 在 GPU 上执行变换失败。
`MCFFT_SETUP_FAILED`	mcFFT 库初始化失败。

3.9.2 Function mcfftExecR2C() 和 mcfftExecD2Z()

```
mcfftResult
      mcfftExecR2C(mcfftHandle plan, mcfftReal *idata, mcfftComplex
↪*odata);
mcfftResult
      mcfftExecD2Z(mcfftHandle plan, mcfftDoubleReal *idata,
↪mcfftDoubleComplex *odata);
```

`mcfftExecR2C()` （`mcfftExecD2Z()`）函数运行单精度 (双精度) 实数到复数的正向 mcFFT 变换。mcFFT 使用由 `idata` 参数指向的 GPU 内存作为输入数据。该函数将非冗余的傅里叶系数存储在 `odata` 数组中。在单精度变换中 `idata` 和 `odata` 指针都需要对齐到 `mcfftComplex` 数据类型，而在双精度变换中则需要对齐到 `mcfftDoubleComplex` 数据类型。如果 `idata` 和 `odata` 相同，`mcfftExecR2C()` (`mcfftExecD2Z()`) 将执行原地变换。正如 `Parameter mcfftType` 中所述，请注意在原地变换和非原地变换之间存在数据布局的差异。

输入

`plan`	由 `mcfftCreate` 返回的 `mcfftHandle`。
`idata`	指向要进行变换的实数输入数据 (存储在 GPU 内存中) 的指针。
`odata`	指向复数输出数据 (存储在 GPU 内存中) 的指针。

输出

odata	包含复数傅里叶系数。

返回值

MCFFT_SUCCESS	mcFFT 成功执行了 FFT 计划。
MCFFT_INVALID_PLAN	plan 参数不是一个有效句柄。
MCFFT_INVALID_VALUE	idata，odata 中至少存在一个无效参数。
MCFFT_INTERNAL_ERROR	检测到内部驱动程序错误。
MCFFT_EXEC_FAILED	mcFFT 在 GPU 上执行变换失败。
MCFFT_SETUP_FAILED	mcFFT 库初始化失败。

3.9.3 Function mcfftExecC2R() 和 mcfftExecZ2D()

```
mcfftResult
      mcfftExecC2R(mcfftHandle plan, mcfftComplex *idata, mcfftReal
↪*odata);
mcfftResult
      mcfftExecZ2D(mcfftHandle plan, mcfftDoubleComplex *idata,
↪mcfftDoubleReal *odata);
```

mcfftExecC2R() （mcfftExecZ2D()）函数运行单精度 (双精度) 实数到复数的逆向 mcFFT 变换。mcFFT 使用由 idata 参数指向的 GPU 内存作为输入数据。输入数组仅包含非冗余的复数傅里叶系数。该函数将实数输出值存储在 odata 数组中。并且在单精度变换中指针需要对齐到 mcfftComplex 数据类型，而在双精度变换中指针则需要对齐到 mcfftDoubleComplex 数据类型。如果 idata 和 odata 相同，mcfftExecC2R() (mcfftExecZ2D()) 将执行原地变换。

输入

plan	由 mcfftCreate 返回的 mcfftHandle。
idata	指向要进行变换的复数输入数据 (存储在 GPU 内存中) 的指针。
odata	指向实数输出数据 (存储在 GPU 内存中) 的指针。

输出

odata	包含实数输出数据。

返回值

MCFFT_SUCCESS	mcFFT 成功执行了 FFT 计划。
MCFFT_INVALID_PLAN	plan 参数不是一个有效句柄。
MCFFT_INVALID_VALUE	idata，odata 中至少存在一个无效参数。
MCFFT_INTERNAL_ERROR	检测到内部驱动程序错误。
MCFFT_EXEC_FAILED	mcFFT 在 GPU 上执行变换失败。
MCFFT_SETUP_FAILED	mcFFT 库初始化失败。

3.10 Function mcfftSetStream()

```
mcfftResult
      mcfftSetStream(mcfftHandle plan, mcStream_t stream);
```

将 MXMACA 流与 mcFFT 计划相关联。现在在计划执行期间的所有内核启动都将通过与之相关联的流进行，从而实现与其他流 (例如数据复制) 的并行。这种关联会一直保持直到计划被销毁或者有其它程序调用另一个 `mcfftSetStream()` 更改流为止。

输入

`plan`	与流关联的 `mcfftHandle` 对象
`stream`	使用 `mcStreamCreate()` 创建一个有效的 MXMACA 流; 默认流 `0`。

状态返回

`MCFFT_SUCCESS`	表示流已与计划关联。
`MCFFT_INVALID_PLAN`	`plan` 参数不是一个有效句柄。

3.11 Function mcfftGetVersion()

```
mcfftResult
      mcfftGetVersion(int *version);
```

返回 mcFFT 的版本号。

输入

`version`	指向版本号的指针。

输出

`version`	包含版本号的变量。

返回值

`MCFFT_SUCCESS`	mcFFT 成功返回了版本号。

3.12 Function mcfftGetProperty()

```
mcfftResult
      mcfftGetProperty(mcfftLibraryPropertyType type, int *value);
```

将动态链接的 mcFFT 库的 `type` 属性数值返回到 `*value` 中。

输入

`type`	mcFFT 库的属性。

输出

`value`	包含所请求属性的整数值。

返回值

MCFFT_SUCCESS	成功返回了属性值。
MCFFT_INVALID_TYPE	未识别到属性类型。
MCFFT_INVALID_VALUE	value 值是 NULL。

3.13 mcFFT 类型

3.13.1 参数 mcfftType

mcFFT 库支持复数数据和实数数据的变换。mcfftType 数据类型是 mcFFT 支持的变换数据类型的枚举。

```
typedef enum mcfftType_t {
      MCFFT_R2C = 0x2a,   // 实数到复数 （交错）
      MCFFT_C2R = 0x2c,   // 复数 （交错） 到实数
      MCFFT_C2C = 0x29,   // 复数到复数 （交错）
      MCFFT_D2Z = 0x6a,   // 双精度到双精度复数 （交错）
      MCFFT_Z2D = 0x6c,   // 双精度复数 （交错） 到双精度
      MCFFT_Z2Z = 0x69    // 双精度复数到双精度复数 （交错）
} mcfftType;
```

3.13.2 变换方向的参数

mcFFT 库根据复数指数项的符号定义正向和逆向的快速傅里叶变换。

```
#define MCFFT_FORWARD -1
#define MCFFT_INVERSE 1
```

mcFFT 执行非归一化 FFT：也就是说在输入数据集上执行正向 FFT，然后在输出数据集上执行逆向 FFT 就可得到与输入相等的数据。用户可以根据数据集大小的倒数对任一变换进行缩放操作，缩放操作通过除以元素的数量进行缩放。

3.13.3 其他 mcFFT 类型

3.13.3.1 mcfftHandle

这是一个用于存储和访问 mcFFT 计划的句柄类型。用户在创建 mcFFT 计划后会获得一个句柄并且需要使用该句柄来执行计划。

```
typedef unsigned int mcfftHandle;
```

3.13.3.2 mcfftReal

单精度浮点实数数据类型。

```
typedef float mcfftReal;
```

3.13.3.3 mcfftDoubleReal

双精度浮点实数数据类型。

```
typedef double mcfftDoubleReal;
```

3.13.3.4 mcfftComplex

单精度浮点复数数据类型由交错的实部和虚部组成。

```
typedef mcComplex mcfftComplex;
```

3.13.3.5 mcfftDoubleComplex

双精度浮点复数数据类型由交错的实部和虚部组成。

```
typedef mcDoubleComplex mcfftDoubleComplex;
```

3.13.3.6 mcfftLibraryPropertyType

`mcfftLibraryPropertyType` 数据类型是库属性类型的枚举。如，mcFFT 版本 X.Y.Z 将枚举产生 `MCFFT_VER_MAJOR=X`，`MCFFT_VER_MINOR=Y`，`MCFFT_VER_PATCH=Z`。

```
typedef enum mcfftLibraryPropertyType_t
{
   MCFFT_VER_MAJOR,
   MCFFT_VER_MINOR,
   MCFFT_VER_PATCH
} mcfftLibraryPropertyType;
```

3.14 常见的类型

3.14.1 mcComplex

用于表示具有单精度实部与虚部的复数的类。拥有两个 float 成员变量 x 和 y，分别表示复数的实部和虚部。

3.14.2 mcDoubleComplex

用于表示具有双精度实部与虚部的复数的类。拥有两个 double 成员变量 x 和 y，分别表示复数的实部和虚部。