傅里叶变换

傅里叶变换（Fourier transform，简称 FT）是一种线性积分变换，用于函数（应用上称作“信号”）在时域和频域之间的变换。因其基本思想首先由法国学者约瑟夫·傅里叶系统地提出，所以以其名字来命名以示纪念。

（连续）傅里叶变换的定义如下：

$$\hat{f}(\xi) = \int_{-\infty}^{\infty} e^{-i2\pi\xi x} f(x) dx$$

其中，f为原函数，$\hat{f}$是原函数的傅里叶变换，x表示时间，其定义域为时域，ξ表示频率，其定义域为频域，i是虚数单位。通常情况下，f是一个实函数，而$\hat{f}$则是一个复数值函数。

离散傅里叶变换

离散傅里叶变换（Discrete Fourier Transform，简称 DFT），是傅里叶变换在时域和频域上都离散的形式，将信号的时域采样变换为频域采样。

对于N点序列$\{x[n]\}_{0\le n<N}$，它的傅里叶变换（DFT）是一个同样长度为N的序列：

$$\hat{x}[k] = \sum_{n=0}^{N-1} e^{-i2\pi\frac{kn}{N}} x[n]$$

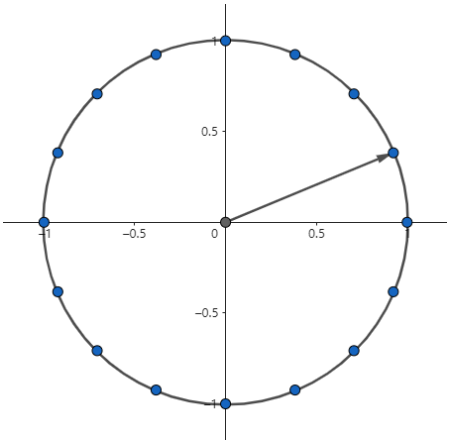
其中e为自然底数，i是虚数单位。通常以符号$\mathcal{F}$表示这一变换，即：

$$\hat{x} = \mathcal{F}x$$

快速傅里叶变换

快速傅里叶变换（Fast Fourier Transform, FFT），是快速计算序列的离散傅里叶变换（DFT）或其逆变换的方法。FFT 能够将计算 DFT 的复杂度从$O(n^2)$降低到 $O\big(nlog(n)\big)$，其中n为数据大小。

在介绍快速傅里叶变换原理之前，我们先引入N次单位根。我们设$\omega_N^k = e^{i2\pi\frac{k}{N}} = \cos\left(2\pi\cdot\frac{k}{N}\right) + i\cdot\sin(2\pi\cdot\frac{k}{N})$，其几何意义是复平面上以原点为起点、单位圆的$N$等分点为终点的向量。特别的，$\omega_N^1$被称作$N$次单位根。



ω_N^k有以下性质：

$$\omega_N^k = (\omega_N^1)^k$$
$$\omega_N^k = \omega_N^{k-1}\cdot\omega_N^1$$
$$\omega_{2N}^{2k} = \omega_N^k$$

$$\omega_N^{k+\frac{N}{2}} = -\omega_N^k$$

$$\omega_N^k = \overline{\omega_N^{-k}}$$

若N为偶数，使用ω_N^k的性质，我们可以进行如下推导:

$$\hat{x}[k] = \sum_{n=0}^{N-1} e^{-i2\pi\frac{kn}{N}} x[n]$$

$$= \sum_{n=0}^{N-1} \omega_N^{-nk} x[n]$$

$$= \sum_{n=0}^{\frac{N}{2}-1} \omega_N^{-2nk} x[2n] + \sum_{n=0}^{\frac{N}{2}-1} \omega_N^{-(2n+1)k} x[2n+1]$$

$$= \sum_{n=0}^{\frac{N}{2}-1} \omega_N^{-2nk} x[2n] + \omega_N^{-k} \sum_{n=0}^{\frac{N}{2}-1} \omega_N^{-2nk} x[2n+1]$$

$$= \sum_{n=0}^{\frac{N}{2}-1} \omega_{\frac{N}{2}}^{-nk} x[2n] + \omega_N^{-k} \sum_{n=0}^{\frac{N}{2}-1} \omega_{\frac{N}{2}}^{-nk} x[2n+1]$$

对于$x\left[k+\frac{N}{2}\right]$，有:

$$\hat{x}\left[k+\frac{N}{2}\right] = \sum_{n=0}^{\frac{N}{2}-1} \omega_{\frac{N}{2}}^{-n\left(k+\frac{N}{2}\right)} x[2n] + \omega_N^{-\left(k+\frac{N}{2}\right)} \sum_{n=0}^{\frac{N}{2}-1} \omega_{\frac{N}{2}}^{-n\left(k+\frac{N}{2}\right)} x[2n+1]$$

$$= \sum_{n=0}^{\frac{N}{2}-1} \omega_{\frac{N}{2}}^{-nk-n\frac{N}{2}} x[2n] + \omega_N^{-k-\frac{N}{2}} \sum_{n=0}^{\frac{N}{2}-1} \omega_{\frac{N}{2}}^{-nk-n\frac{N}{2}} x[2n+1]$$

$$= \sum_{n=0}^{\frac{N}{2}-1} \omega_{\frac{N}{2}}^{-nk} x[2n] - \omega_N^{-k} \sum_{n=0}^{\frac{N}{2}-1} \omega_{\frac{N}{2}}^{-nk} x[2n+1]$$

因此只需要求得$\sum_{n=0}^{\frac{N}{2}-1} \omega_{\frac{N}{2}}^{-nk} x[2n]$以及$\sum_{n=0}^{\frac{N}{2}-1} \omega_{\frac{N}{2}}^{-nk} x[2n+1]$，即可求得$\hat{x}[k]$以及$\hat{x}\left[k+\frac{N}{2}\right]$。

而求$\sum_{n=0}^{\frac{N}{2}-1} \omega_{\frac{N}{2}}^{-nk} x[2n]$以及$\sum_{n=0}^{\frac{N}{2}-1} \omega_{\frac{N}{2}}^{-nk} x[2n+1]$的过程相当于对$x$的奇数下标部分和偶数下标部分分别做离散傅里叶变换。

通过一次将原本的计算分解为奇数部分和偶数部分的操作，我们将原本离散傅里叶变换中计算所有 $\hat{x}[k]$所需的N^2次乘法分解为2个向量长度为$\frac{N}{2}$的需要$\frac{N^2}{4}$次乘法的更小的离散傅里叶变换，继续地分解下去直到向量的长度为 1，可以将总的计算复杂度缩减到$Nlog(N)$。

一维快速傅里叶变换的一种基础实现方式的伪代码如下所示:

算法：$FFT(A,N,w)$

输入：时域信号，长度为N的向量A；信号长度，2的幂N；N次单位根w。

输出：频域信号，长度为N的向量

if $N = 1$ ***then***

    return A

else

    $Aeven \leftarrow Vector(\ [A[1], A[3], \dots, A[N-1]]\ )$

    $Aodd \leftarrow Vector(\ [A[2], A[4], \dots, A[N]]\ )$

    $Veven \leftarrow FFT(\ Aeven, N/2, w\text{^}2\ )$

    $Vodd \leftarrow FFT(\ Aodd, N/2, w\text{^}2\ )$

    $V \leftarrow Vector(N)$ #定义一个长度为 N 的向量

    for $i = 1\ to\ N/2$ ***do***

        $V[i] \leftarrow Veven[i] + w\text{^}(i-1) * Vodd[i]$

        $V[N/2 + i] \leftarrow Veven[i] - w\text{^}(i-1) * Vodd[i]$

    end do

    return V

end if

实数的一维离散傅里叶变换

若对于$\forall n \in [0, N-1]$，有$x[n] \in R$，那么

$$\hat{x}[0] = \sum_{n=0}^{N-1} \omega_N^0 x[n] = \sum_{n=0}^{N-1} x[n]$$

所以$\hat{x}[0]$一定是实数。

此外，还有：

$$\hat{x}[N-k] = \sum_{n=0}^{N-1} \omega_N^{-n(N-k)} x[n]$$

$$= \sum_{n=0}^{N-1} \omega_N^{-nN+nk} x[n]$$

$$= \sum_{n=0}^{N-1} \omega_N^{nk} x[n]$$

$$= \sum_{n=0}^{N-1} \overline{\omega_N^{-nk}} x[n]$$

$$= \overline{\sum_{n=0}^{N-1} \omega_N^{-nk} x[n]}$$

$$= \overline{\hat{x}[k]}$$

所以，当x序列为实数序列时，$\hat{x}[k]=\overline{\hat{x}[N-k]}$，即$\hat{x}[k]$与$\overline{\hat{x}[N-k]}$共轭。例如当 N=7 时，$\hat{x}[1]=\overline{\hat{x}[6]}$，$\hat{x}[2]=\overline{\hat{x}[5]}$，$\hat{x}[3]=\overline{\hat{x}[4]}$；当$N=8$时，$\hat{x}[1]=\overline{\hat{x}[7]}$，$\hat{x}[2]=\overline{\hat{x}[6]}$，$\hat{x}[3]=\overline{\hat{x}[5]}$，$\hat{x}[4]=\overline{\hat{x}[4]}$。

在进行实数的快速傅里叶变换时，考虑到输入的实数数组和输出的复数数组所占的空间大小不一致，为了能够将输出结果放在原数组当中（这种输出方法叫做 in place），Intel FFT 库和 FFTW 利用了结果的对称共轭的性质，比如对于一维 FFT，只输出前$\frac{N}{2}+1$个结果。此外，由于当x序列为实数序列时，$\hat{x}[0]$和$\hat{x}\left[\frac{N}{2}\right]$（若$N$为偶数）的虚部一定为 0，一些方法还会将其虚部省略从而进一步地缩减输出结果所需的内存空间。

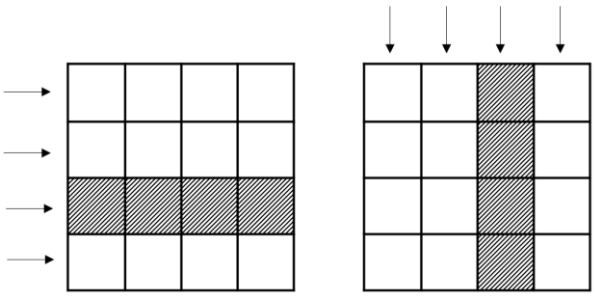
二维离散傅里叶变换

一般做影像处理的影像大多不是连续信号，而对于平面上的不连续信号，我们则需使用二维离散傅立叶变换。

假设输入的影像为x，水平方向长度是N，垂直方向长度是M，二维离散傅立叶变换定义为：

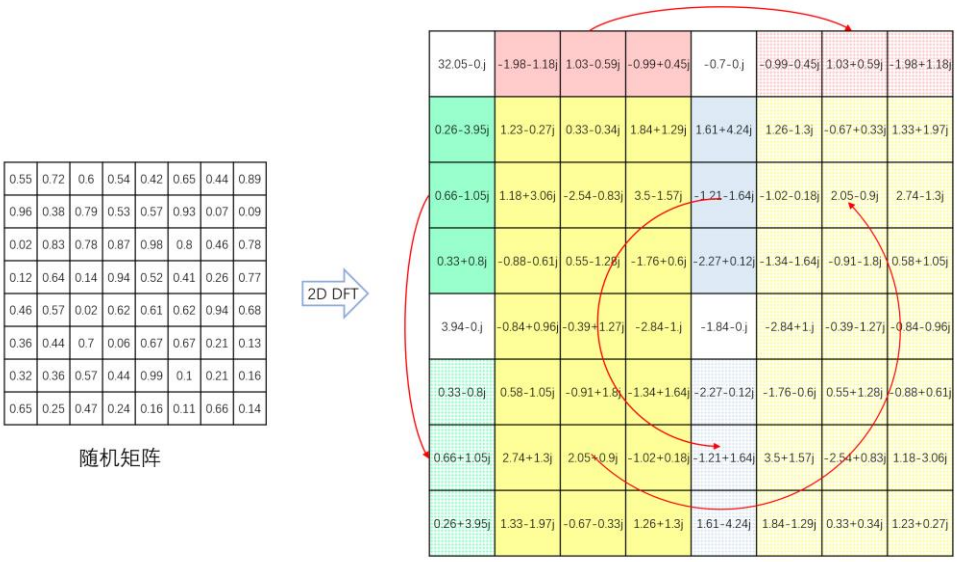
$$\hat{x}[u,v]=\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}e^{-i2\pi(\frac{um}{M}+\frac{vn}{N})}x[m,n]$$

二维离散傅里叶变换的基本实现方法是先对每一行进行一次一维离散傅里叶变换之后，再对每一列进行一次一维离散傅里叶变换。



实数的二维离散傅里叶变换

就像实数的一维离散傅里叶变换一样，实数的二维离散傅里叶变换结果也会呈现出一种特殊的共轭对称性，oneMKL FFT 库和 FFTW 也利用了这一特性来节约输出序列所占的空间。



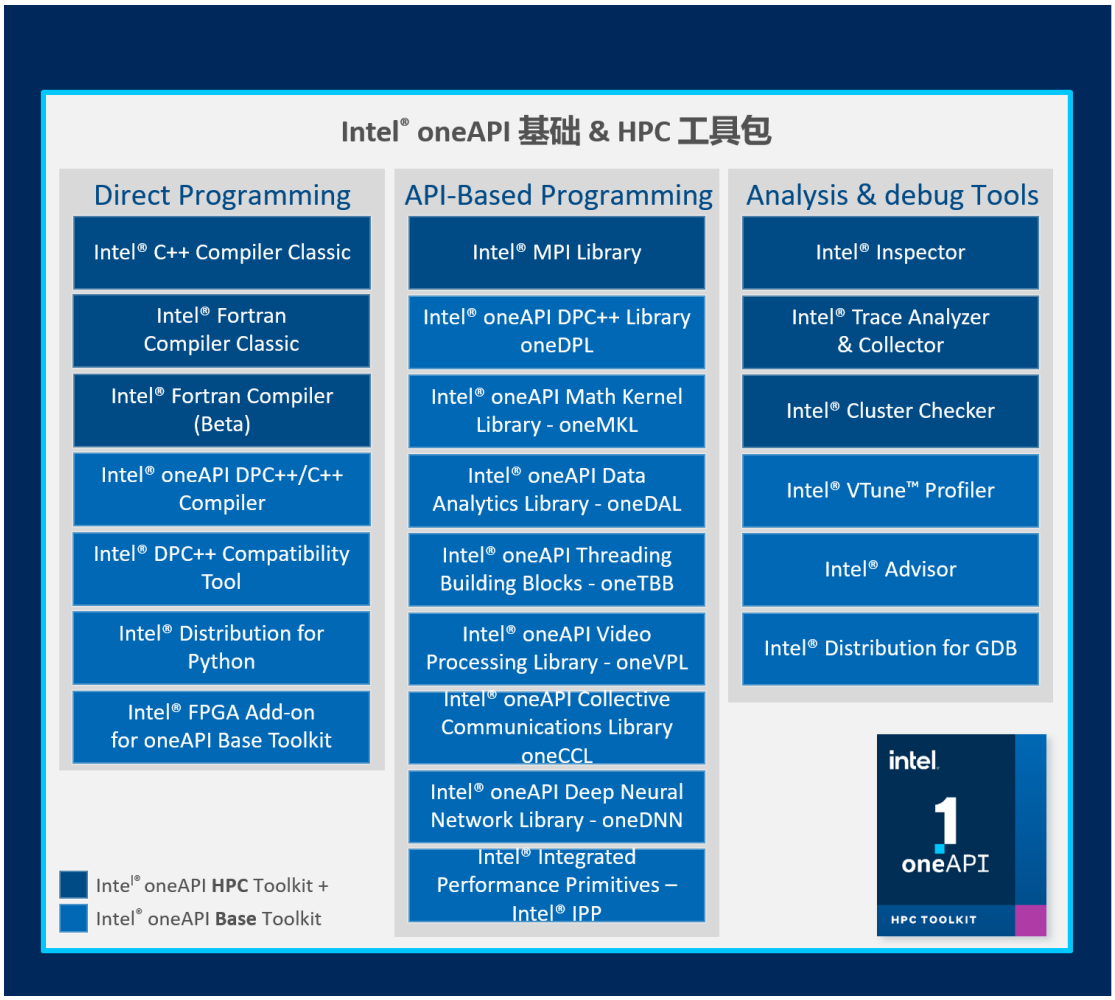
随机矩阵

上图展示了当$N = 8$时，随机实数矩阵进行二维离散傅里叶变换后所得到的结果。图右结果矩阵中，实心背景部分与相同颜色虚线背景部分使用了箭头指示，可以看出相对应的部分呈现对称共轭关系。

oneAPI 介绍

oneAPI 是一个开源规范，旨在简化希望创建基于加速器的应用程序和希望支持各种硬件架构和硬件供应商的开发人员的生活。

oneAPI Base Toolkits 和 HPC Toolkits 的内容如下图所示：



oneMKL 下载安装

英特尔®oneAPI 数学内核库（Intel® oneAPI Math Kernel Library，简称 oneMKL）是一个计算数学库，其中包含高度优化的广泛线程例程，适用于需要最高性能的应用程序。该库提供 Fortran 和 C 编程语言接口。oneMKL C 语言接口可以从用 C 或 C++以及可以引用 C 接口的任何其他语言编写的应用程序中调用。

oneMKL 在这些主要计算领域提供全面的功能支持：

- BLAS（1 级、2 级和 3 级）和 LAPACK 线性代数例程，提供向量、向量矩阵和矩阵矩阵运算。
- ScaLAPACK 分布式处理线性代数例程，以及基本线性代数通信子程序(BLACS)和并行基本线性代数子程序(PBLAS)。
- oneMKL PARDISO（基于并行直接稀疏求解器 PARDISO*的直接稀疏求解器），迭代稀疏求解器，支持用于求解稀疏方程组的稀疏 BLAS（1、2 和 3 级）例程，以及分布式版本提供 oneMKL PARDISO 求解器用于集群。
- 一维、二维或三维中的快速傅立叶变换(FFT)函数，支持混合基数（不限于 2 的幂的大小），以及提供在集群上使用的这些函数的分布式版本。

- 用于优化向量数学运算的向量数学(VM)例程。
- 矢量统计(VS)例程，为多种概率分布、卷积和相关例程以及汇总统计函数提供高性能矢量化随机数生成器(RNG)。
- 数据拟合库，提供基于样条的函数逼近、函数导数和积分以及搜索功能。
- 扩展特征求解器，基于 Feast 特征值求解器的特征求解器的共享内存编程(SMP)版本。

英特尔® oneAPI 数学核心库(oneMKL)针对英特尔处理器上的性能进行了优化。oneMKL 还可以在非 Intel x86 兼容处理器上运行。

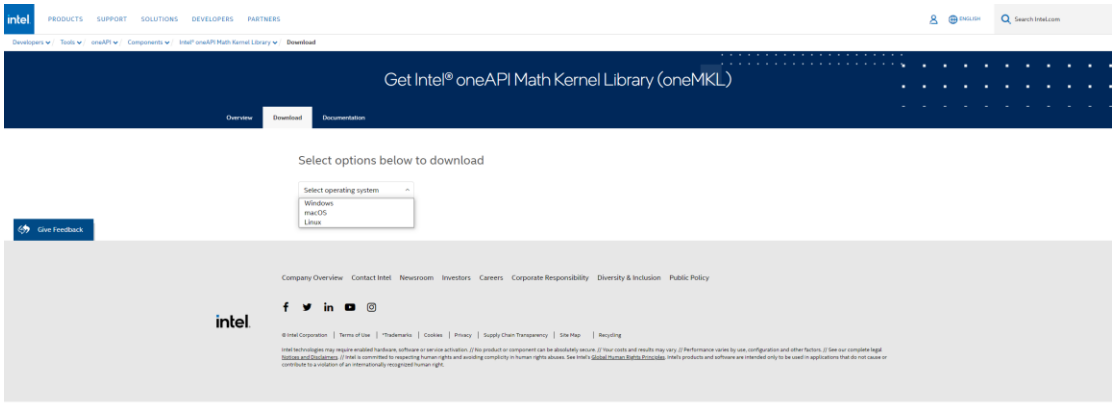
oneMKL 可以作为单独的软件包下载：

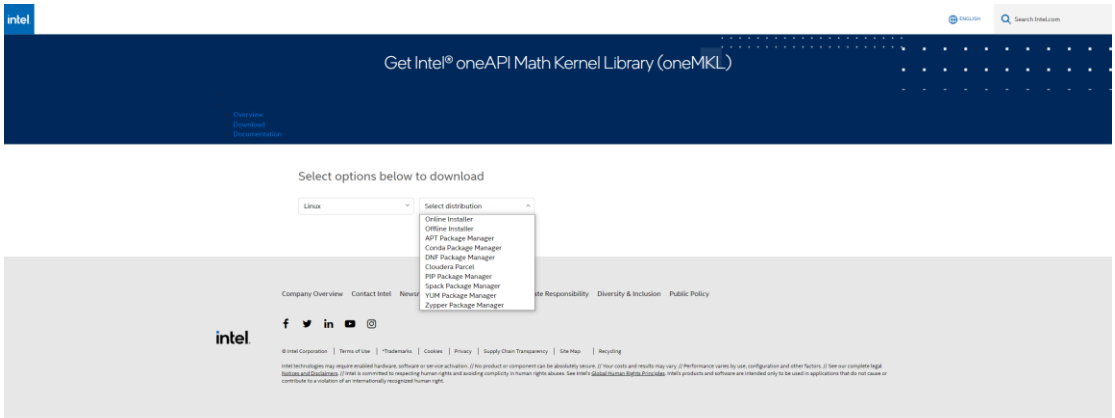
https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl-download.html

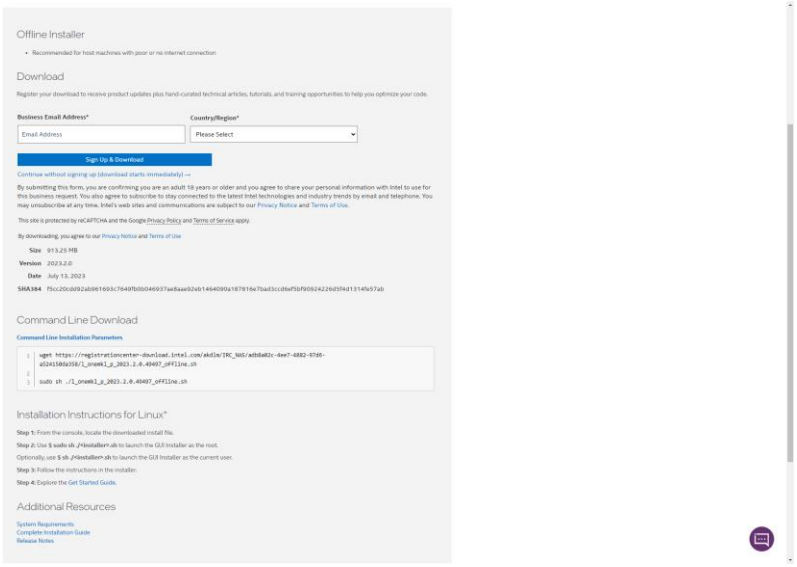
也可以作为 Intel® oneAPI Base Toolkit 的一部分下载：

https://www.intel.com/content/www/us/en/developer/tools/oneapi/base-toolkit-download.html

进入到下载界面后，选择自己的操作系统以及在线或者离线安装方式之后，按照提示进行安装即可。







例如，这里我们选择了 Linux 的离线安装方式，只需执行以下命令即可：

```
wget https://registrationcenter-
download.intel.com/akdlm/IRC_NAS/adb8a02c-4ee7-4882-97d6-
a524150da358/l_onemkl_p_2023.2.0.49497_offline.sh
sudo sh ./l_onemkl_p_2023.2.0.49497_offline.sh
```

安装脚本./l_onemkl_p_2023.2.0.49497_offline.sh 的常用参数如下：

参数		含义
-h 或--help		显示帮助
-s 或--silent		无屏幕输出安装
-a <arguments>	-c 或--cli	使用文本用户界面（TUI）模式来交互安装
	--eula accept/decline	在静默模式下支持，接受终端用户协议（EULA）
	--install-dir	在静默模式下支持，自定义安装目录
	-s 或--silent	以静默模式运行安装程序

例如：

```
# 显示安装程序帮助：
./l_onemkl_p_2023.2.0.49497_offline.sh -a -h
# 显示可用的产品：
./l_onemkl_p_2023.2.0.49497_offline.sh -a --list-products
# 以 GUI 模式运行安装程序：
./l_onemkl_p_2023.2.0.49497_offline.sh
# 以命令行界面运行安装程序：
./l_onemkl_p_2023.2.0.49497_offline.sh -a -c
# 以静默模式运行安装程序：
./l_onemkl_p_2023.2.0.49497_offline.sh -a -s --eula accept
```

oneMKL 随机数生成器

oneMKL 随机数生成器的典型流程如下：

1. 创建和初始化流。函数 vslNewStream, vslNewStreamEx, vslCopyStream, vslCopyStreamState, vslLeapfrogStream, vslSkipAheadStream, vslSkipAheadStreamEx。

2. 调用一个或多个随机数生成器。
3. 处理输出。
4. 使用 vslDeleteStream 函数删除一个或多个流。

生成单精度随机数的示例代码如下：

```
#include <stdio.h>
#include "mkl_vsl.h"
float r[1000]; /* buffer for random numbers */
VSLStreamStatePtr stream;
/* Initializing */
vslNewStream(&stream, VSL_BRNG_MT19937, 777);
/* Generating */
vsRngUniform(VSL_RNG_METHOD_UNIFORM_STD, stream, 1000, r, 0.0, 1.0);
/* Deleting the stream */
vslDeleteStream(&stream);
```

oneMKL FFT 函数

oneMKL 提供了一个接口，用于通过快速傅里叶变换算法计算离散傅里叶变换。函数名称中的前缀 Dfti 和配置参数名称中的前缀 DFTI 代表离散傅立叶变换接口。

oneMKL 中提供的快速傅里叶变换函数有以下 2 种实现：

1. 适用于单处理器或共享内存系统的 FFT 函数
2. 适用于分布式内存架构的 Cluster FFT 函数（仅适用于 Intel® 64 架构）

FFT 和 Cluster FFT 函数均通过五个步骤计算 FFT：

1. 通过调用 DftiCreateDescriptor 或 DftiCreateDescriptorDM 函数为问题分配新的描述符。描述符记录变换的配置，例如维度（或等级）、大小、变换数量、输入/输出数据的内存布局（由步幅定义）和缩放因子。许多配置设置都在此调用中分配了默认值，您可能需要在应用程序中修改这些默认值。
2. （可选）根据需要调用 DftiSetValue 或 DftiSetValueDM 函数来调整描述符配置。通常，您必须仔细定义 FFT 的数据存储布局或集群 FFT 进程之间的数据分布。描述符的配置设置（例如默认值）可以使用 DftiGetValue 或 DftiGetValueDM 函数获取。
3. 通过调用 DftiCommitDescriptor 或 DftiCommitDescriptorDM 函数来提交描述符，也就是说，使描述符为变换计算做好准备。一旦描述符被提交，变换的参数，例如变换的类型和数量、步幅和距离、数据的类型和存储布局等等，就被“冻结”在描述符中。
4. 根据需要多次调用 DftiComputeForward / DftiComputeBackward 或 DftiComputeForwardDM / DftiComputeBackwardDM 函数来计算变换。由于描述符是单独定义和提交的，因此计算函数所做的就是获取输入和输出数据并按照定义计算变换。要修改对计算函数的另一次调用的任何配置参数，请使用 DftiSetValue 后跟 DftiCommitDescriptor（DftiSetValueDM 后跟 DftiCommitDescriptorDM）或创建并提交另一个描述符。
5. 通过调用 DftiFreeDescriptor 或 DftiFreeDescriptorDM 函数来释放描述符。这会将描述符内部消耗的内存返回给操作系统。

所有上述函数都返回一个整数状态值，成功完成操作后该值为零。您可以借助 DftiErrorClass 或 DftiErrorMessage 函数解释非零状态。

使用 oneMKL FFT 实现单精度二维的复数到复数和实数到复数原地快速傅里叶变换的代码示例如下：

```
/* C99 example */
#include "mkl_dfti.h"
/* complex data in, complex data out */
float _Complex c2c_data[32][100];
/* real data in, complex data out */
float r2c_data[34][102];
DFTI_DESCRIPTOR_HANDLE my_desc1_handle = NULL;
DFTI_DESCRIPTOR_HANDLE my_desc2_handle = NULL;
MKL_LONG status;MKL_LONG dim_sizes[2] = {32, 100};
/* ...put values into c2c_data[i][j] 0<=i<=31, 0<=j<=99 */
/* ...put values into r2c_data[i][j] 0<=i<=31, 0<=j<=99 */
status = DftiCreateDescriptor(&my_desc1_handle, DFTI_SINGLE,
DFTI_COMPLEX, 2, dim_sizes);
status = DftiCommitDescriptor(my_desc1_handle);
status = DftiComputeForward(my_desc1_handle, c2c_data);
status = DftiFreeDescriptor(&my_desc1_handle);
/* result is the complex value c2c_data[i][j], 0<=i<=31, 0<=j<=99 */
status = DftiCreateDescriptor(&my_desc2_handle, DFTI_SINGLE,
DFTI_REAL, 2, dim_sizes);
status = DftiCommitDescriptor(my_desc2_handle);
status = DftiComputeForward(my_desc2_handle, r2c_data);
status = DftiFreeDescriptor(&my_desc2_handle);
/* result is the complex r2c_data[i][j] 0<=i<=31, 0<=j<=99
and is stored in CCS format*/
```

实数的快速傅里叶变换，为了能够实现原地变换，oneMKL FFT 不会将完整的结果写入数组当中，默认的 CCE 格式的布局如下图所示，在默认的行间距 DFTI_OUTPUT_STRIDES 设置下，下图中结果矩阵中的有色色块将会被连续地写入结果数组中，空白背景的部分将不会被写入结果数组当中：

0.55	0.72	0.6	0.54	0.42	0.65	0.44	0.89
0.96	0.38	0.79	0.53	0.57	0.93	0.07	0.09
0.02	0.83	0.78	0.87	0.98	0.8	0.46	0.78
0.12	0.64	0.14	0.94	0.52	0.41	0.26	0.77
0.46	0.57	0.02	0.62	0.61	0.62	0.94	0.68
0.36	0.44	0.7	0.06	0.67	0.67	0.21	0.13
0.32	0.36	0.57	0.44	0.99	0.1	0.21	0.16
0.65	0.25	0.47	0.24	0.16	0.11	0.66	0.14

随机矩阵

2D DFT

32.05-0.j	-1.98-1.18j	1.03-0.59j	-0.99+0.45j	-0.7-0.j	-0.99-0.45j	1.03+0.59j	-1.98+1.18j
0.26-3.95j	1.23-0.27j	0.33-0.34j	1.84+1.29j	1.61+4.24j	1.26-1.3j	-0.67+0.33j	1.33+1.97j
0.66-1.05j	1.18+3.06j	-2.54-0.83j	3.5-1.57j	-1.21-1.64j	-1.02-0.18j	2.05-0.9j	2.74-1.3j
0.33+0.8j	-0.88-0.61j	0.55-1.28j	-1.76+0.6j	-2.27+0.12j	-1.34-1.64j	-0.91-1.8j	0.58+1.05j
3.94-0.j	-0.84+0.96j	-0.39+1.27j	-2.84-1.j	-1.84-0.j	-2.84+1.j	-0.39-1.27j	-0.84-0.96j
0.33-0.8j	0.58-1.05j	-0.91+1.8j	-1.34+1.64j	-2.27-0.12j	-1.76-0.6j	0.55+1.28j	-0.88+0.61j
0.66+1.05j	2.74+1.3j	2.05+0.9j	-1.02+0.18j	-1.21+1.64j	3.5+1.57j	-2.54+0.83j	1.18-3.06j
0.26+3.95j	1.33-1.97j	-0.67-0.33j	1.26+1.3j	1.61-4.24j	1.84-1.29j	0.33+0.34j	1.23+0.27j

我们如果需要获取完整的结果数组，可以使用如下示例的代码进行转换：

```
float* r2c_data;// malloc(sizeof(float)*(N1+2)*(N2+2))
// on input : R{k1, k2} = r2c_data[N2*k1]
status = DftiComputeForward(r2c_data); // real-to-complex FFT
// on output : Z{k1, k2} = re + I*im, where
if(k1==0)
{
    if(k2<(N2/2+1))
    {
        re = r2c_data[k2*2];
        im = r2c_data[k2*2+1];
    }
    else
    {
        re = r2c_data[(N2-k2)*2];
        im = -r2c_data[(N2-k2)*2+1];
    }
}
else
{
    if(k2<(N2/2+1))
    {
        re = r2c_data[(k1*(N2/2+1)+k2)*2];
        im = r2c_data[(k1*(N2/2+1)+k2)*2+1];
    }
    else
    {
        re = r2c_data[((N1-k1)*(N2/2+1)+(N2-k2))*2];
        im = -r2c_data[((N1-k1)*(N2/2+1)+(N2-k2))*2+1];
```

```
    }
}
```

使用 oneMKL Cluster FFT 实现双精度二维的复数到复数异地快速傅里叶变换的代码示例如下:

```
/* C99 example */
#include "mpi.h"
#include "mkl_cdft.h"
DFTI_DESCRIPTOR_DM_HANDLE desc = NULL;
MKL_LONG v, i, j, n, s;
Complex *in, *out;
MKL_LONG dim_sizes[2] = {nx, ny};
MPI_Init(...);
/* Create descriptor for 2D FFT */
DftiCreateDescriptorDM(MPI_COMM_WORLD,
&desc, DFTI_DOUBLE, DFTI_COMPLEX, 2, dim_sizes);
/* Ask necessary length of in and out arrays and allocate memory */
DftiGetValueDM(desc,CDFT_LOCAL_SIZE,&v);
in = (Complex*) malloc(v*sizeof(Complex));
out = (Complex*) malloc(v*sizeof(Complex));
/* Fill local array with initial data. Current process performs n rows,
0 row of in corresponds to s row of virtual global array */
DftiGetValueDM(desc, CDFT_LOCAL_NX, &n);
DftiGetValueDM(desc, CDFT_LOCAL_X_START, &s);
/* Virtual global array globalIN is defined by function f as
globalIN[i*ny+j]=f(i,j) */
for(i = 0; i < n; ++i)
for(j = 0; j < ny; ++j) in[i*ny+j] = f(i+s,j);
/* Set that we want out-of-place transform (default is DFTI_INPLACE) */
DftiSetValueDM(desc, DFTI_PLACEMENT, DFTI_NOT_INPLACE);
/* Commit descriptor, calculate FFT, free descriptor */
DftiCommitDescriptorDM(desc);
DftiComputeForwardDM(desc, in, out);
/* Virtual global array globalOUT is defined by function g as
globalOUT[i*ny+j]=g(i,j) Now out contains result of FFT.
out[i*ny+j]=g(i+s,j) */
DftiFreeDescriptorDM(&desc);
free(in);
free(out);
MPI_Finalize();
```

oneMKL FFTW 包装器

oneMKL 提供 FFTW2 和 FFTW3 接口到 oneMKL FFT 和三角变换功能。这些接口的目的是让使用 FFTW (www.fftw.org)的应用程序在不改变程序源代码的情况下获得 oneMKL 的

性能。

FFTW2 和 FFTW3 接口都是作为 oneMKL 的包装器开源提供的。为了便于使用, FFTW3 接口也集成在 oneMKL 中。

使用 FFTW3 接口实现单精度实数到复数原地快速傅里叶变换的代码示例如下:

```
#include "fftw3.h"
int N1 = 10;
int N2 = 15;
float* x = 0;
x = fftwf_malloc(sizeof(float)*2*N1*(N2/2+1));
r2c = fftwf_plan_dft_r2c_2d(N1, N2, x, (fftwf_complex*)x,
FFTW_ESTIMATE);
fftwf_execute(r2c);
fftwf_destroy_plan(r2c);
fftwf_free(x);
```