Convolution

A good exercise is to write a circular shift using FFT, first on 1D arrays and then on 2D arrays.

The shift theorem

Multiplying x_n by a linear phase $e^{\frac{2\pi i}{N}nm}$ for some integer m corresponds to a circular shift of the output X_k: X_k is replaced by X_{k-m}, where the subscript is interpreted modulo N (i.e., periodically). Similarly, a circular shift of the input x_n corresponds to multiplying the output X_k by a linear phase. Mathematically, if x_n represents the vector x then

$\texttt{if}\ \mathcal{F}(\{x_n\})_k = X_k$
$\texttt{then}\ \mathcal{F}(\{x_n \cdot e^{\frac{2\pi i}{N}nm}\})_k = X_{k-m}$
$\texttt{and}\ \mathcal{F}(\{x_{n-m}\})_k = X_k \cdot e^{-\frac{2\pi i}{N}km}$

CUDA Fortran code implementing the task

```
module mulvv_m
  use precision_m
contains
  attributes (global) subroutine mulvv(c, a, b, n)
    implicit none
    complex(fp_kind), intent(out) :: c(*)
    complex(fp_kind), intent(in) :: a(*), b(*)
    !integer, intent(in) :: n
    integer, value :: n
    integer :: i

    i = blockDim % x *( blockIdx %x -1) + threadIdx % x
    if ( i<=n ) c(i) = a(i) * b(i)
  end subroutine mulvv

end module mulvv_m
```

```
program conv1DTest
  use precision_m
  use cufft_m
  use mulvv_m

  implicit none

  integer :: tPB = 512

  integer, parameter :: n=8
  integer :: plan, planType, i

  complex(fp_kind) :: a(n),b(n), kern(n)
  complex(fp_kind), device :: a_d(n), b_d(n), kern_d(n)

  real:: pi=4._fp_kind*atan(1._fp_kind)
```

program conv1DTest (2)

```
print *, pi

!initialize arrays on host
do i=1,n
 a(i) = cmplx(real(i,fp_kind),0._fp_kind)
end do

!kernel
do i=1,n
kern(i) = cmplx(cos(2._fp_kind*pi/n*3._fp_kind*(i-1)),
                sin(2._fp_kind*pi/n*3._fp_kind*(i-1)))
end do
kern_d = kern
```

```
!copy arrays to device
a_d = a

! Print initial array
print *, "Array A:"
print *, a

! set planType to either single or double precision
if (fp_kind == singlePrecision) then
   planType = CUFFT_C2C
else
   planType = CUFFT_Z2Z
endif
```

program conv1DTest (4)

```
! initialize the plan and execute the FFTs.

call cufftPlan1D(plan,n,planType,1)
call cufftExec(plan,planType,a_d,b_d,CUFFT_FORWARD)

call mulvv<<<ceiling(real(n)/tPB ), tPB>>>
                        (b_d, b_d, kern_d, n)

!!  !$cuf kernel do <<<*,*>>>
!!   do i=1,n
!!    b_d(i) = b_d(i)*kern_d(i)
!!   end do

call cufftExec(plan,planType,b_d,b_d,CUFFT_INVERSE)


! Copy results back to host
```

Output from the program:

```
   3.141593
 Array A:
 (1.000000,0.000000)  (2.000000,0.000000)  (3.000000,0.0000
 (4.000000,0.000000)  (5.000000,0.000000)  (6.000000,0.0000
 (7.000000,0.000000)  (8.000000,0.000000)
 Array B
 (3.999999,-4.3510994E-07)  (4.999999,5.8412155E-07)  (5.99
  (7.000000,1.2397727E-06)  (8.000001,5.1856438E-07)  (1.00
   (2.000001,-1.1503657E-06)  (3.000000,-1.2636224E-06)
```

so the circular shift:

```
4 -> 5 -> 6 -> 7 -> 8 -> 1 -> 2 -> 3
^                                  ^
|    . . .                         |
1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8
```

CUF kernels

CUF kernel is an pseudo-commentary - a loop annotation from which compiler automatically generates CUDA kernel.

```
!! call mulvv<<<ceiling(real(n)/tPB ), tPB>>>(b_d, b_d, ker

  !$cuf kernel do <<<*,*>>>
   do i=1,n
     b_d(i) = b_d(i)*kern_d(i)
   end do
```