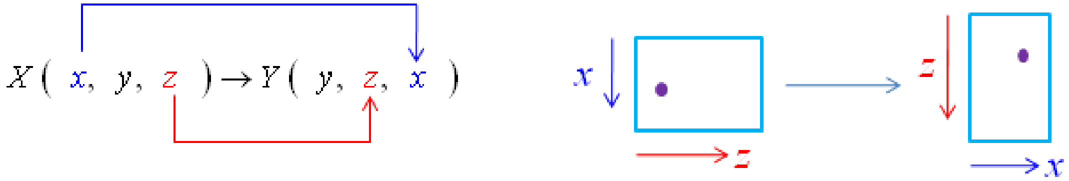
Matrix transpose

Author : Lung-Sheng Chien

National Tsing Hua university, R.O.C (Taiwan)

Mail: d947207@oz.nthu.edutw

Abstract: we provide 2D transpose and 3D transpose in this document, source code of 2D transpose comes from SDK and then we use the same idea to build 3D transpose of the form



Experiment shows that our 3D result has “good” performance comparable with 2D result (2D transpose is optimized version)

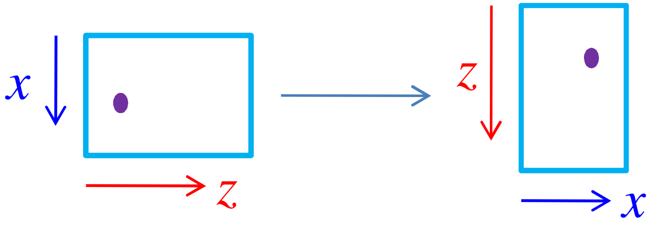
Transpose on 3D data

Objective: given 3D data $X(1:n_1, 1:n_2, 1:n_3)$, we want to do transpose operation under $(y, z, x) \leftarrow (x, y, z)$

such that $X(1:n_1, 1:n_2, 1:n_3) \to Y(1:n_2, 1:n_3, 1:n_1)$ with utilization of coalesce property.

Observation: it is similar to 2D transpose, if we only consider x-z slice when fixed y

$$X(\ x,\ \ y,\ \ z\ \ ) \to Y(\ \ y,\ \ z,\ \ x\ \ )$$



The simplest way is (1) use 2D grid to represent (x, z) slice and do transpose operation along y

```
for $y = 1:n_2$
    for each threads in $x$-$z$ slice, do transpose
    $X(idx, y, idz) \to Y(y, idz, idx)$
endfor
```

→ we must use share memory to decrease latency

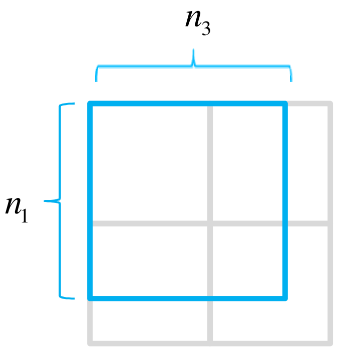
Remark: motivation of transpose on 3D data comes from 3D FFT (see Sine_transform_3D.ppt)

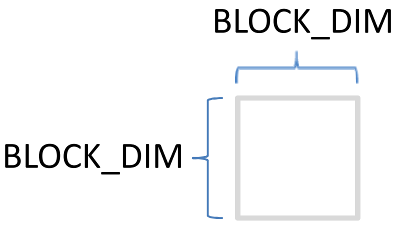
Transpose 3D: framework [1]

Objective: define a framework to do xyz2yzx transpose operation and use different kind of techniques

The number of grids in z-direction to cover x-z slice is (n3 + BLOCK_DIM-1)/ BLOCK_DIM

The number of grids in x-direction to cover x-z slice is (n1 + BLOCK_DIM-1)/ BLOCK_DIM

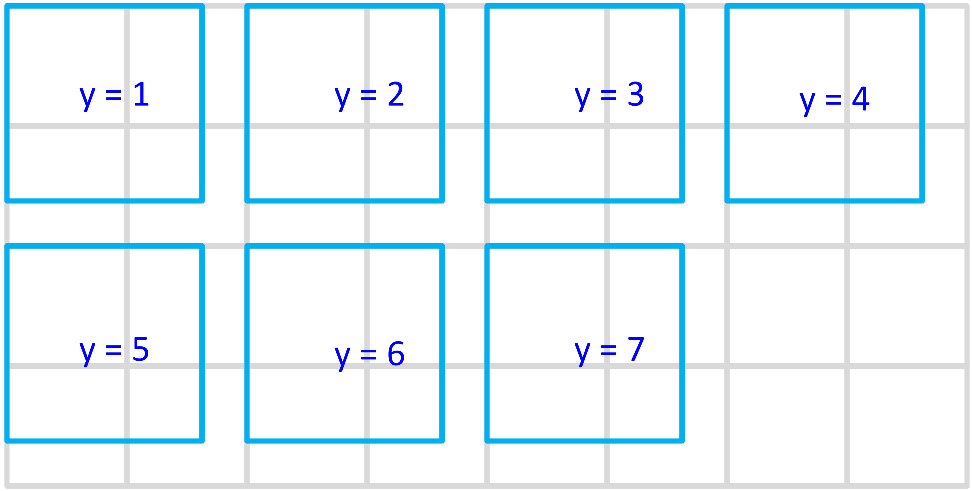




$$Gx \equiv \frac{n_1 + BLOCK\_DIM - 1}{BLOCK\_DIM} = 2$$

$$Gz \equiv \frac{n_3 + BLOCK\_DIM - 1}{BLOCK\_DIM} = 2$$

Assume n2 = 7 (7 x-z slice), then what is configuration of grid?



$$k_1 \equiv floor\left(\sqrt{n_2}\right), \quad k_2 \equiv ceil\left(\frac{n_2}{k_1}\right)$$

$$then \quad grid = \left(k_2 \cdot Gz, k_1 \cdot Gx\right)$$

$$\Big\downarrow \; n_2 = 7$$

$$k_1 = floor\left(\sqrt{7}\right) = 2, \quad k_2 = ceil\left(\frac{7}{2}\right) = 4$$

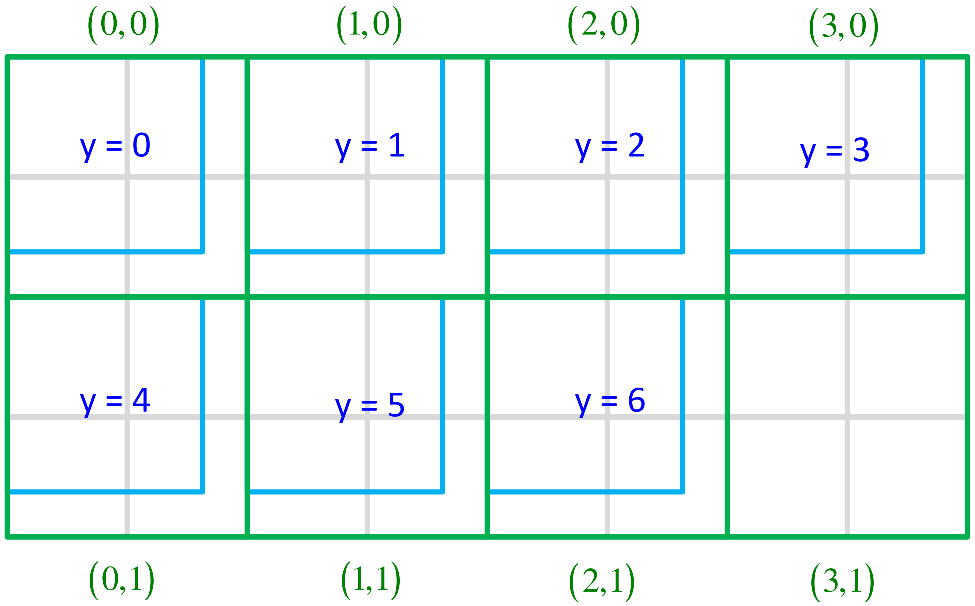
Transpose 3D: framework [2]

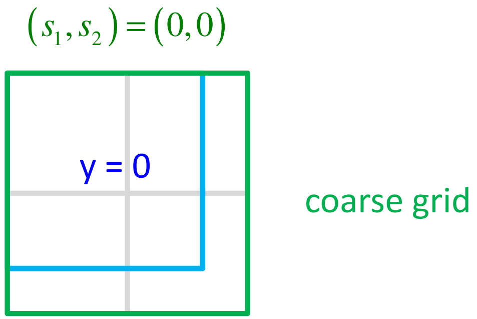
Trick: we can do better grid configuration such that resource utilization is highest

```
for $k_1 = floor\left(\sqrt{n_2}\right) : -1 : 1$
    $k_2 = ceil\left(\frac{n_2}{k_1}\right)$
    if $k_1 k_2 - n_2 \le 1$, then $grid = \left(k_2 \cdot Gz, k_1 \cdot Gx\right)$, break
end
```

Then we waste one block at most

Express $blockIdx.x = Gz \cdot s_1 + t_1$, $blockIdx.y = Gx \cdot s_2 + t_2$ such that (s_1, s_2): index to y-direction, (t_1, t_2): index to x - z slice where $s_1 = floorf\left(blockIdx.x / Gz\right)$, $t_1 = blockIdx.x - Gz \cdot s_1$





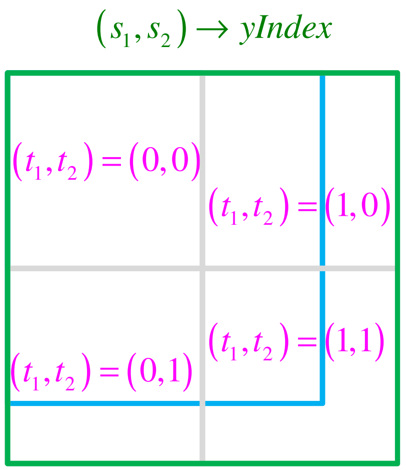
$yIndex = s_2 k_2 + s_1$

where $grid = \left(k_2 \cdot Gz, k_1 \cdot Gx\right)$

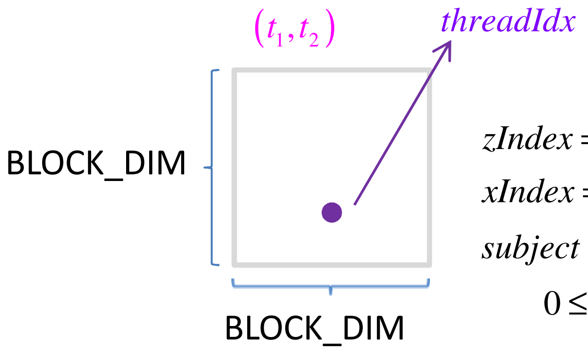
$k_2 = \#$ of coarse grid along z

$= gridDim.x / Gz$

Transpose 3D: framework [3]



(t_1,t_2) may be regarded as local block ID



$$zIndex = t_1 \cdot BLOCK\_DIM + threadIdx.x$$

$$xIndex = t_2 \cdot BLOCK\_DIM + threadIdx.y$$

$$subject \quad (zIndex, xIndex) \equiv (k-1, i-1) \quad \text{since } X(1:n_1, 1:n_2, 1:n_3)$$

$$0 \le xIndex < n_1 \quad , \quad 0 \le zIndex < n_3$$

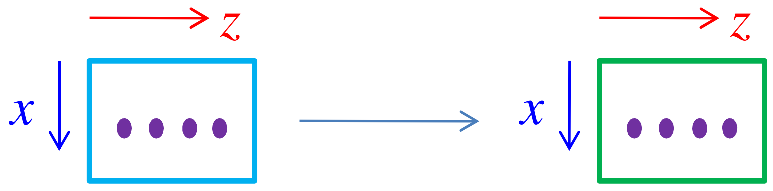
1 Fixed y, real (x, y, z) to shared memory

$$row-major(i,j,k) = (i-1)n_2 n_3 + (j-1)n_3 + (k-1)$$

```
for  $i = 1 : n_1$
    for  $k = 1 : n_3$
        $shared[i][k] \leftarrow X(i,j,k)$
    endfor
endfor
```

$$\xrightarrow[\substack{xIndex = i-1 \\ yIndex = j-1 \\ zIndex = k-1}]{X(1:n_1, 1:n_2, 1:n_3)}$$

```
if  $(xIndex < n_1)$  and  $(zIndex < n_3)$
    $index\_in = xindex \cdot n_2 n_3 + yIndex \cdot n_3 + zIndex$
    $shared[threadIdx.y][threadIdx.x] := X[index\_in]$
endif
__syncthreads( )
```

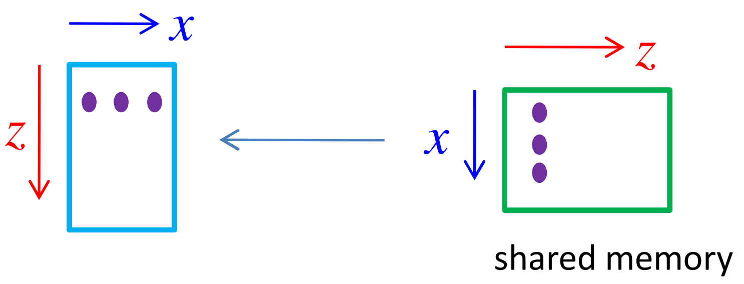


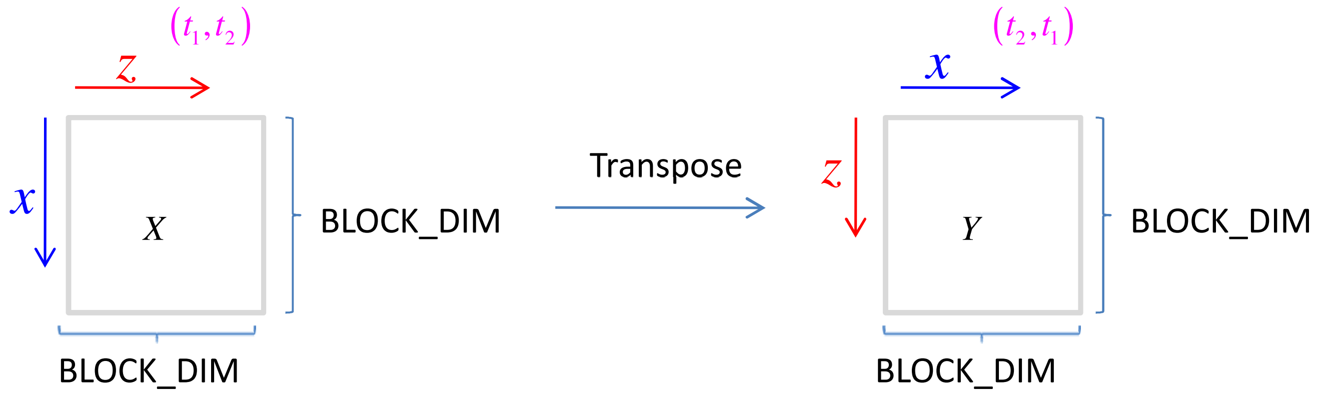
shared memory

Transpose 3D: framework [4]

2 Transpose data in shared memory to $Y(1:n_2,1:n_3,1:n_1)$

```
$for\ \ k = 1:n_3$
    $for\ \ i = 1:n_1$
        $Y(j,k,i) \leftarrow shared[i][k]$
    $endfor$
$endfor$
```





$$zIndex = t_1 \cdot BLOCK\_DIM + threadIdx.x$$
$$xIndex = t_2 \cdot BLOCK\_DIM + threadIdx.y$$

$$xIndex = t_2 \cdot BLOCK\_DIM + threadIdx.x$$
$$zIndex = t_1 \cdot BLOCK\_DIM + threadIdx.y$$

$$\xrightarrow[row-major(j,k,i)=(j-1)n_3n_1+(k-1)n_1+(i-1)]{Y(1:n_2,1:n_3,1:n_1)}$$

```
$if\ \ (xIndex < n_1)$ and $(zIndex < n_3)$
    $index\_out = yindex \cdot n_3 n_1 + zIndex \cdot n_1 + xIndex$
    $Y[index\_out] := shared[threadIdx.x][threadIdx.y]$
$endif$
```