Usage of high-performance libraries for GPUs

IOANNIS E. VENETIS

Assistant Professor
Department of Informatics
University of Piraeus, Greece

Achieving high performance on GPUs

Modern GPUs have the potential of achieving high performance

- ≈33.5 TFLOPS for Double Precision arithmetic (H100 and H200 GPUs)
- ≈90.5 TFLOPS for Single Precision arithmetic (L40 GPU)
- ≈989.4 TFLOPS for Half Precision arithmetic (H100 GPU)

Programming in a GPU hardware-agnostic manner will not unleash the full potential of GPUs

Programming for high performance on a GPU is not an easy task

- Good knowledge of the GPU hardware is required
- Good knowledge of possible performance bottlenecks is required
- Good knowledge of how to map programming constructs to the GPU hardware to avoid bottlenecks is required

Purpose of this presentation

Present highly optimized libraries for GPUs that provide commonly required operations in a range of scientific domains

- cuBLAS, cuSPARSE, CUDA Graphs, cuSOLVER, cuFFT, cuRAND, AmgX, …

https://developer.nvidia.com/gpu-accelerated-libraries

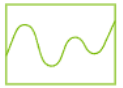
CUDA Math Libraries

GPU-accelerated math libraries lay the foundation for compute-intensive applications in areas such as molecular dynamics, computational fluid dynamics, computational chemistry, medical imaging, and seismic exploration.

cuBLAS

GPU-accelerated basic linear algebra (BLAS) library.

Learn More >



cuFFT

GPU-accelerated library for Fast Fourier Transform implementations.

Learn More >

cuRAND

GPU-accelerated random number generation.

Learn More >

cuSOLVER

GPU-accelerated dense and sparse direct solvers.

Learn More >

cuSPARSE

GPU-accelerated BLAS for sparse matrices.

Learn More >

cuTENSOR

GPU-accelerated tensor linear algebra library.

Learn More >

cuDSS

GPU-accelerated direct sparse solver library.

Learn More >

CUDA Math API

GPU-accelerated standard mathematical function APIs.

Learn More >

AmgX

GPU-accelerated linear solvers for simulations and implicit unstructured methods.

Learn More >

Basic Linear Algebra Subprograms (BLAS)

What are BLAS?

A specification that prescribes a set of low-level routines for performing common linear algebra operations

- Vector addition, scalar multiplication, dot products, linear combinations, matrix multiplication, …

They are the de facto standard low-level routines for linear algebra libraries

BLAS functionality is categorized into three sets of routines called “levels”

- Level 1: vector operations
  - Dot products, vector norms, generalized vector addition, …
- Level 2: matrix-vector operations and triangular solve
  - Matrix-vector multiplication, …
- Level 3: matrix-matrix operations and triangular solve with multiple right-hand sides
  - Generalized matrix-matrix multiplication, …

BLAS assumes dense matrices

cuBLAS

cuBLAS

An implementation of BLAS from NVidia for its GPUs

- Web site: https://developer.nvidia.com/gpu-accelerated-libraries
- Documentation: https://docs.nvidia.com/cuda/cublas/index.html

Provides four sets of APIs:

- cuBLAS (starting with CUDA 6.0)
  - Common API for most uses, single GPU
- cuBLASXt (starting with CUDA 6.0)
  - Execution of BLAS routines on multi-GPUs settings
- cuBLASLt (starting with CUDA 10.1)
  - Lightweight library dedicated to GEneral Matrix-matrix Multiply (GEMM) operations
- cuBLASDx (in preview stage, 03/2025)
  - Provides functionality for performing BLAS calculations inside CUDA kernels
  - Fusion of numerical operations can decrease latency and improve performance

Matrices can be either zero-base indexed or one-base indexed

cuBLAS

A set of routines are called from host code (but involve the GPU)

- Helper routines:
  - Memory allocation
  - Data copying from CPU to GPU, and vice versa
  - Error reporting
- Compute routines:
  - e.g. matrix-matrix, matrix-vector, …

Some calls are asynchronous

- The routine starts executing on the GPU
- The host code continues execution before the routine on the GPU finishes

Error handling

CUDA

- Data type: cudaError_t
- Call to CUDA routine returns a value of the above type
  - cudaSuccess: Call to routine completed successfully
  - Otherwise, routine completed with an error

cuBLAS

- Data type: cublasStatus_t
- Call to cuBLAS routine returns a value of the above type
  - CUBLAS_STATUS_SUCCESS : Call to routine completed successfully
  - Otherwise, routine completed with an error

cuBLAS library context

Data type cublasHandle_t provides a handle to the cuBLAS library context

Function cublasCreate() must be called to initialize a handle

- cublasDestroy() to destroy the handle

The handle is explicitly passed to every subsequent library function call

```
cublasStatus_t cublasStat;
cublasHandle_t handle;
…
cublasStat = cublasCreate(&handle);
if (cublasStat != CUBLAS_STATUS_SUCCESS) {
  printf("cuBLAS initialization failed\n");
  return EXIT_FAILURE;
}
```

Some Level 1 cuBLAS functions

Function	Operation
cublasI<t>amax()	Finds the (smallest) index of the element of the maximum magnitude
cublasI<t>amin()	Finds the (smallest) index of the element of the minimum magnitude
cublasI<t>asum()	Computes the sum of the absolute values of the elements of a vector
cublasI<t>dot	Computes the dot product of two vectors
cublas<t>nrm2()	Computes the Euclidean norm of a vector
cublas<t>rot()	Applies Givens rotation matrix to two vectors

Some Level 2 cuBLAS functions

Function	Operation
cublas<t>gemv()	General matrix-vector multiplication
cublas<t>gbmv()	Banded matrix-vector multiplication
cublas<t>symv()	Symmetric matrix-vector multiplication
cublas<t>sbmv()	Symmetric banded matrix-vector multiplication
cublas<t>tbmv()	Triangular banded matrix-vector multiplication
cublas<t>ger()	Rank-1 update
cublas<t>syr()	Symmetric rank-1 update

Some Level 3 cuBLAS functions

Function	Operation
cublas<t>gemm()	Matrix-matrix multiplication
cublas<t>symm()	Symmetric matrix-matrix multiplication
cublas<t>trmm()	Triangular matrix-matrix multiplication
cublas<t>hemm()	Hermitian matrix-matrix multiplication
cublas<t>syrk()	Symmetric rank-k update

cublas<t>axpy()

```
cublasStatus_t cublas<t>axpy(cublasHandle_t handle, int n,
                             const <t>              *alpha,
                             const <t>              *x, int incx,
                             <t>                    *y, int incy)
```

<t> defines the data type

- S, D: real, Single- or Double-precision
- C, Z: complex, Single- or Double-precision

Performs the operation $y \leftarrow alpha \cdot x + y$

- x and y are vectors of size n
- $alpha$ is scalar

Param.	Meaning
handle	Handle to the cuBLAS library context
n	Number of elements in the vector x and y
alpha	<type> scalar used for multiplication.
x	<type> vector with n elements
incx	Stride between consecutive elements of x
y	<type> vector with n elements
incy	Stride between consecutive elements of y

cublas<t>axpy() example

```
#include <stdio.h>
#include <stdlib.h>
#include <cuda_runtime.h>
#include "cublas_v2.h"
#define n 32

int main()
{
  cudaError_t     cudaStat;
  cublasStatus_t  cublasStat;
  cublasHandle_t  handle;

  int i;
  float *x, *y, *d_x, *d_y, alpha = 1.0;

  x = (float *)malloc(n * sizeof(*x)); // Allocate vectors
  y = (float *)malloc(n * sizeof(*y)); // on host

  for (i = 0; i < n; i++) {            // Initialize vectors
    x[i] = (float)i;
    y[i] = (float)i;
  }

  // Allocate vectors on GPU
  cudaStat = cudaMalloc((void**)&d_x, n * sizeof(*x));
  cudaStat = cudaMalloc((void**)&d_y, n * sizeof(*y));
```

```
  // Create cuBLAS handle
  cublasStat = cublasCreate(&handle);

  // Copy vectors from host to GPU
  cublasStat = cublasSetVector(n, sizeof(*x), x, 1, d_x, 1);
  cublasStat = cublasSetVector(n, sizeof(*y), y, 1, d_y, 1);

  // Perform operation for single precision ('S') operands
  cublasStat = cublasSaxpy(handle, n, &alpha, d_x, 1, d_y, 1);

  // Copy result from GPU to host
  cublasStat = cublasGetVector(n, sizeof(*y), d_y, 1, y, 1);

  printf("y = alpha * x + y\n");
  for (i = 0; i < n; i++) {
    printf("%f\n", y[i]);
  }

  cudaStat   = cudaFree(d_x);
  cudaStat   = cudaFree(d_y);
  cublasStat = cublasDestroy(handle);
  free(x);
  free(y);

  return EXIT_SUCCESS;
}
```

cuBLAS

In addition to the computational functions:

- Support for a single function call to do a “batch” of smaller operations
  - Level 2: cublas<t>gemvBatched(), cublas<t>gemvStridedBatched()
  - Level 3: cublas<t>gemmBatched(), cublas<t>gemmStridedBatched()□, cublas<t>gemmGroupedBatched(), cublas<t>trsmBatched()
  - See also section “BLAS-like Extension” in the documentation
- Support for using CUDA streams to do a large number of small tasks concurrently
- Support for mixed / low precision operations

How to connect the dots

Calling a function in cuBLAS (and other libraries) requires some setup

- i.e., calling additional functions of the library

The documentation only describes each function individually

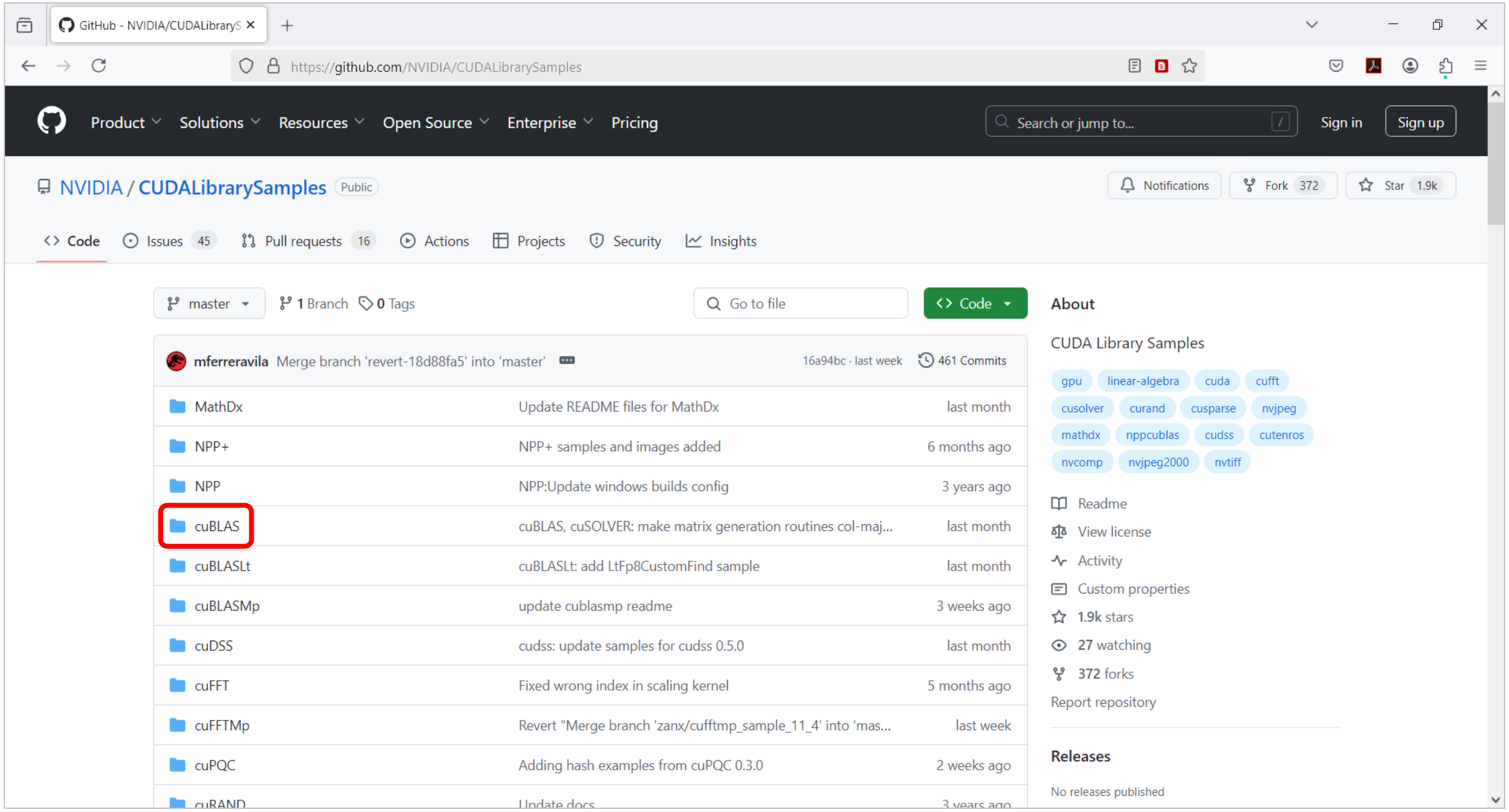
- How do we know what else is required, i.e. how do we get the “great picture”?

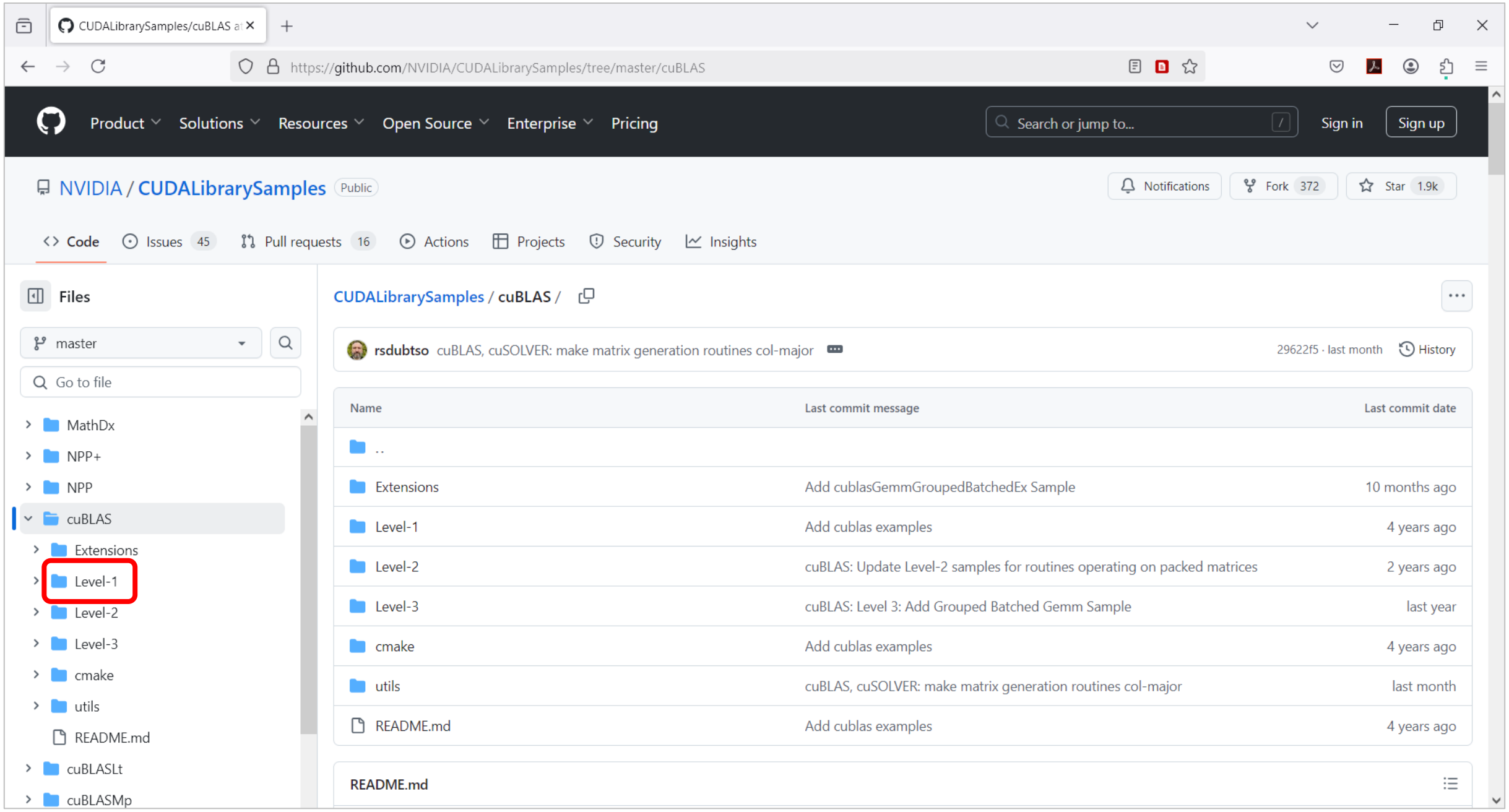
NVidia provides a repository with many examples:

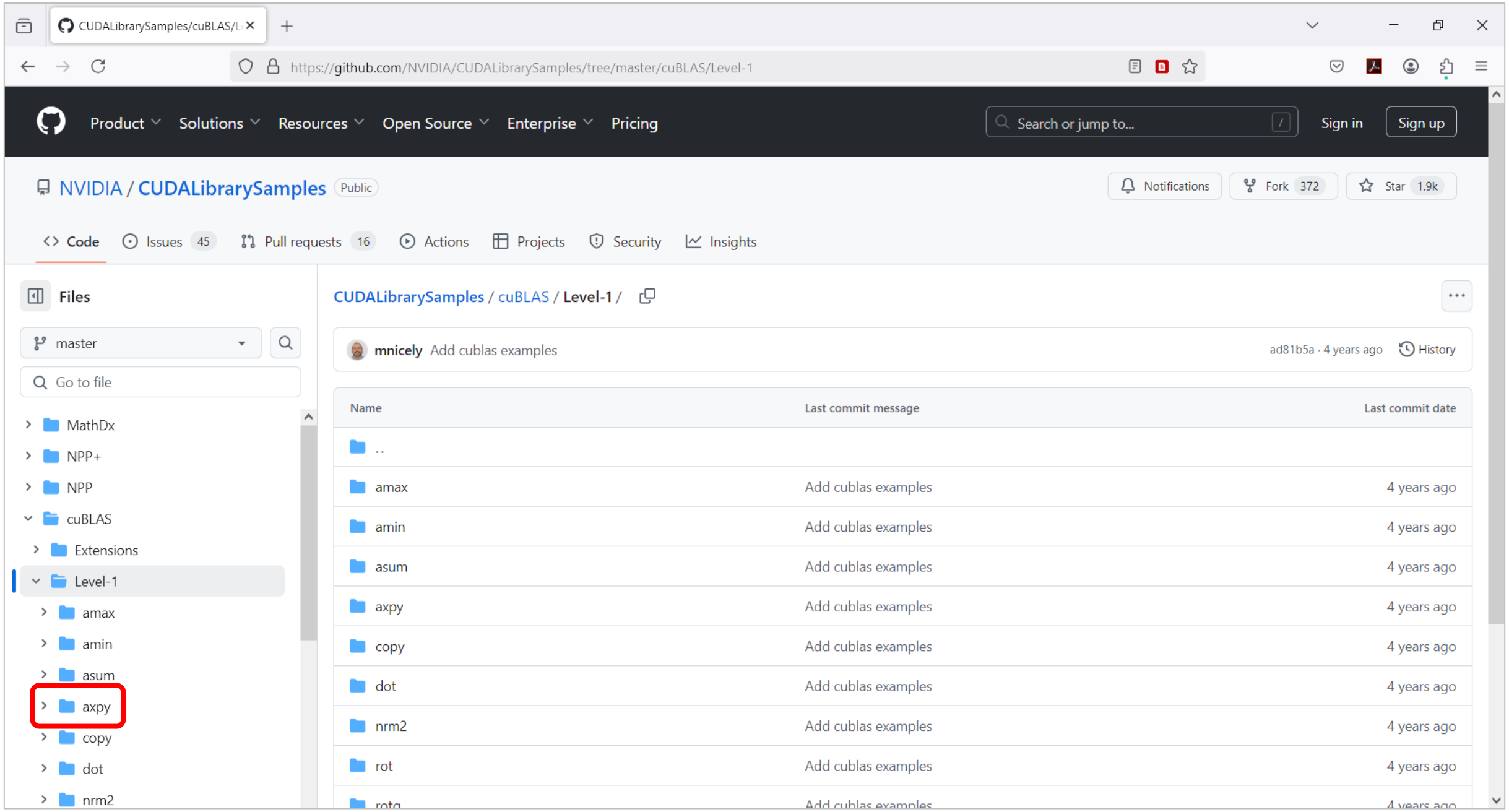
https://github.com/NVIDIA/CUDALibrarySamples

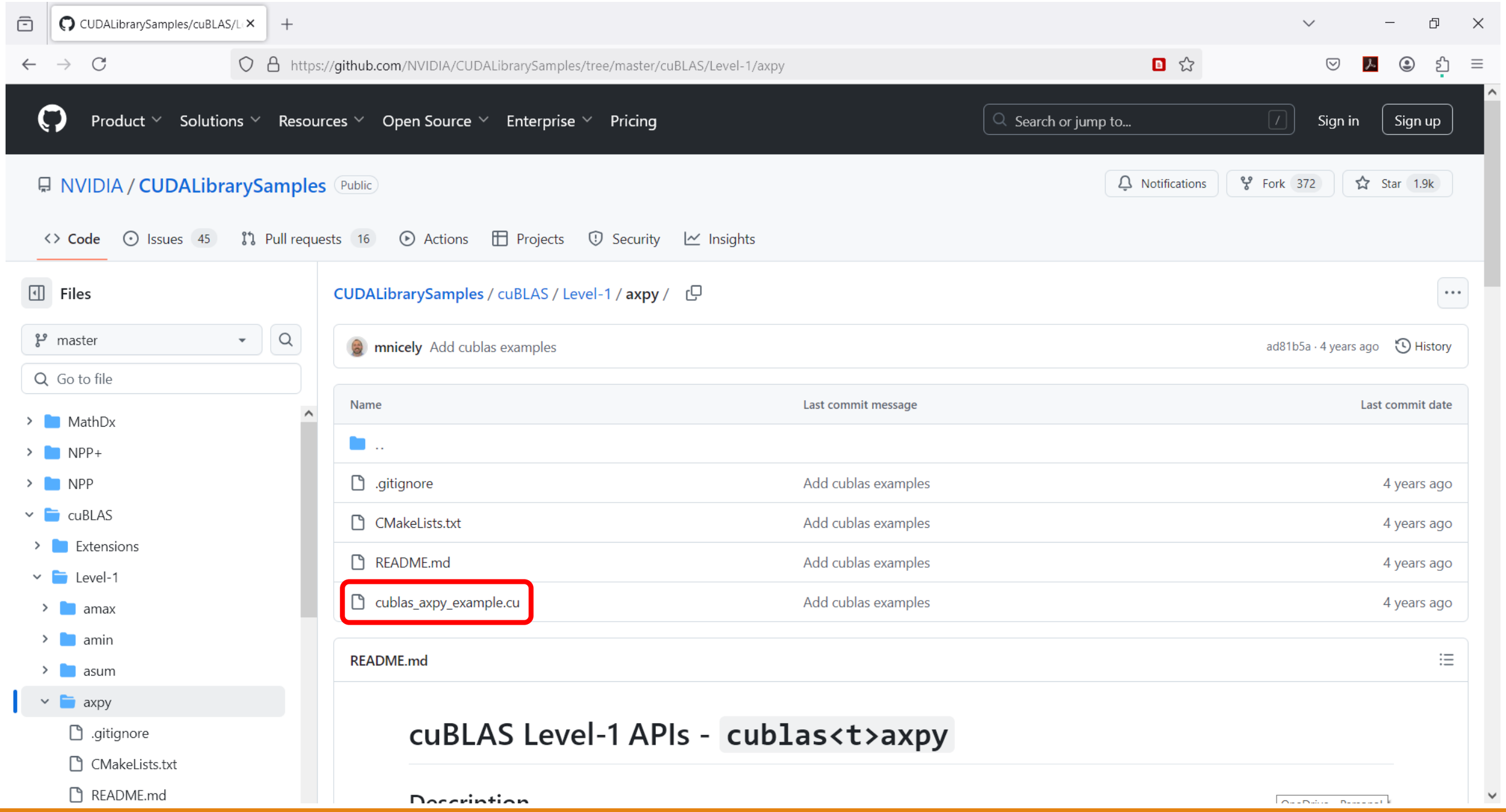
Small, easy to follow, complete, working examples with all required steps included

Well organized per targeted library


GitHub - NVIDIA/CUDALibrarys
https://github.com/NVIDIA/CUDALibrarySamples
Product
Solutions
Resources
Open Source
Enterprise
Pricing
Search or jump to...
Sign in
Sign up
NVIDIA / CUDALibrarySamples
Public
Notifications
Fork 372
Star 1.9k
Code
Issues 45
Pull requests 16
Actions
Projects
Security
Insights
master
1 Branch
0 Tags
Go to file
Code
mferreravila Merge branch 'revert-18d88fa5' into 'master'
16a94bc · last week
461 Commits
MathDx Update README files for MathDx last month
NPP+ NPP+ samples and images added 6 months ago
NPP NPP:Update windows builds config 3 years ago
cuBLAS cuBLAS, cuSOLVER: make matrix generation routines col-maj... last month
cuBLASLt cuBLASLt: add LtFp8CustomFind sample last month
cuBLASMp update cublasmp readme 3 weeks ago
cuDSS cudss: update samples for cudss 0.5.0 last month
cuFFT Fixed wrong index in scaling kernel 5 months ago
cuFFTMp Revert "Merge branch 'zanx/cufftmp_sample_11_4' into 'mas... last week
cuPQC Adding hash examples from cuPQC 0.3.0 2 weeks ago
cuRAND Update docs 3 years ago
About
CUDA Library Samples
gpu linear-algebra cuda cufft
cusolver curand cusparse nvjpeg
mathdx nppcublas cudss cutenros
nvcomp nvjpeg2000 nvtiff
Readme
View license
Activity
Custom properties
1.9k stars
27 watching
372 forks
Report repository
Releases
No releases published


CUDALibrarySamples/cuBLAS at
https://github.com/NVIDIA/CUDALibrarySamples/tree/master/cuBLAS
Product
Solutions
Resources
Open Source
Enterprise
Pricing
Search or jump to...
Sign in
Sign up
NVIDIA / CUDALibrarySamples
Public
Notifications
Fork 372
Star 1.9k
Code
Issues 45
Pull requests 16
Actions
Projects
Security
Insights
Files
master
Go to file
MathDx
NPP+
NPP
cuBLAS
Extensions
Level-1
Level-2
Level-3
cmake
utils
README.md
cuBLASLt
cuBLASMp
CUDALibrarySamples / cuBLAS /
rsdubtso cuBLAS, cuSOLVER: make matrix generation routines col-major
29622f5 · last month
History
Name
Last commit message
Last commit date
..
Extensions
Add cublasGemmGroupedBatchedEx Sample
10 months ago
Level-1
Add cublas examples
4 years ago
Level-2
cuBLAS: Update Level-2 samples for routines operating on packed matrices
2 years ago
Level-3
cuBLAS: Level 3: Add Grouped Batched Gemm Sample
last year
cmake
Add cublas examples
4 years ago
utils
cuBLAS, cuSOLVER: make matrix generation routines col-major
last month
README.md
Add cublas examples
4 years ago
README.md


CUDALibrarySamples/cuBLAS/L
https://github.com/NVIDIA/CUDALibrarySamples/tree/master/cuBLAS/Level-1
Product
Solutions
Resources
Open Source
Enterprise
Pricing
Search or jump to...
Sign in
Sign up
NVIDIA / CUDALibrarySamples
Public
Notifications
Fork 372
Star 1.9k
Code
Issues 45
Pull requests 16
Actions
Projects
Security
Insights
Files
master
Go to file
MathDx
NPP+
NPP
cuBLAS
Extensions
Level-1
amax
amin
asum
axpy
copy
dot
nrm2
CUDALibrarySamples / cuBLAS / Level-1 /
mnicely Add cublas examples
ad81b5a · 4 years ago
History
Name
Last commit message
Last commit date
..
amax Add cublas examples 4 years ago
amin Add cublas examples 4 years ago
asum Add cublas examples 4 years ago
axpy Add cublas examples 4 years ago
copy Add cublas examples 4 years ago
dot Add cublas examples 4 years ago
nrm2 Add cublas examples 4 years ago
rot Add cublas examples 4 years ago


CUDALibrarySamples/cuBLAS/L
https://github.com/NVIDIA/CUDALibrarySamples/tree/master/cuBLAS/Level-1/axpy
Product
Solutions
Resources
Open Source
Enterprise
Pricing
Search or jump to...
Sign in
Sign up
NVIDIA / CUDALibrarySamples
Public
Notifications
Fork 372
Star 1.9k
Code
Issues 45
Pull requests 16
Actions
Projects
Security
Insights
Files
master
Go to file
MathDx
NPP+
NPP
cuBLAS
Extensions
Level-1
amax
amin
asum
axpy
.gitignore
CMakeLists.txt
README.md
CUDALibrarySamples / cuBLAS / Level-1 / axpy /
mnicely Add cublas examples
ad81b5a · 4 years ago
History
Name
Last commit message
Last commit date
..
.gitignore
Add cublas examples
4 years ago
CMakeLists.txt
Add cublas examples
4 years ago
README.md
Add cublas examples
4 years ago
cublas_axpy_example.cu
Add cublas examples
4 years ago
README.md
cuBLAS Level-1 APIs - cublas<t>axpy
Description

Sparse matrices

Special matrices

In many problems of various scientific fields, special matrices occur

Typically, they aresparse matrices

- Many of their elements are zero

Density of sparse matrix

$$Density\ of\ sparse\ matrix = \frac{number\ of\ non-zero\ elements}{number\ of\ all\ elements}$$

Sparse matrices can be divided into two broad categories

- The non-zero elements have some special structure
- The non-zero elements do not have some special structure

We will focus on matrices that do not have some special structure

Special matrices

Goal: Store only non-zero elements of sparse matrices

- Reduces amount of memory required to store the matrix
- Avoids redundant computations with elements that are zero
  - E.g., we know the result of adding zero to a value, or when multiplying a value with zero

Arranging the elements in two dimensions is fine for mathematics, even in the presence of many zeroes

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,m} \end{bmatrix}$$

For programming, it does not satisfy the above goals

Special matrices

Sparse matrices are handled differently than dense matrices in programs

- They are stored more efficiently, depending on their structure
- They are accessed according to the way they have been stored

Algorithms to perform operations on dense matrices do not apply to sparse matrices that have been stored in a more efficient manner

- Sparse Matrix-Vector (SpMV) multiplication, Sparse Matrix-Dense Matrix multiplication, Sparse Matrix-Sparse Matrix multiplication, etc.

Representation of a sparse matrix

Several different formats are used to represent a sparse matrix

- COO: Coordinate format
- CSR: Compressed Sparse Row
- CSC: Compressed Sparse Column
- BSR: Block Sparse Row
- …

Coordinate format (COO)

Simple to understand

Simple to build a sparse matrix

Uses three vectors to store:

- Row
  - Non-zero elements row numbers
- Column
  - Non-zero elements column numbers
- Data
  - Non-zero elements values

Elements can be stored in any order

- Sorting either row or column numbers can improve performance
- If sorting is applied to one vector, elements must be moved accordingly in the other two vectors

Even when sorted, not great for computations

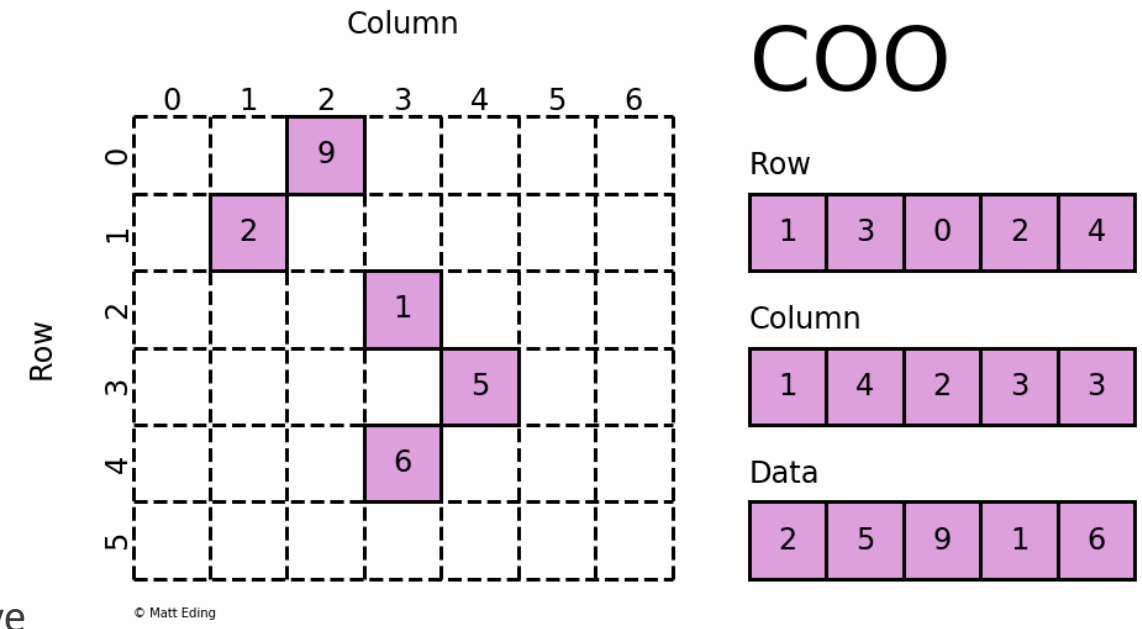


Image from https://matteding.github.io/2019/04/25/sparse-matrices

Compressed Sparse Row format (CSR)

Faster computations

More difficult to understand

Uses three vectors:

- Index Pointers
  - Look at adjacent elements, e.g. call them start, end
    - Position of first element (start) in vector determines row number
    - Range [start:end) determines range in vector Indices for current row
    - end – start is the number of non-zero elements in the current row
- Indices
  - Columns of non-zero elements
- Data
  - Non-zero elements values

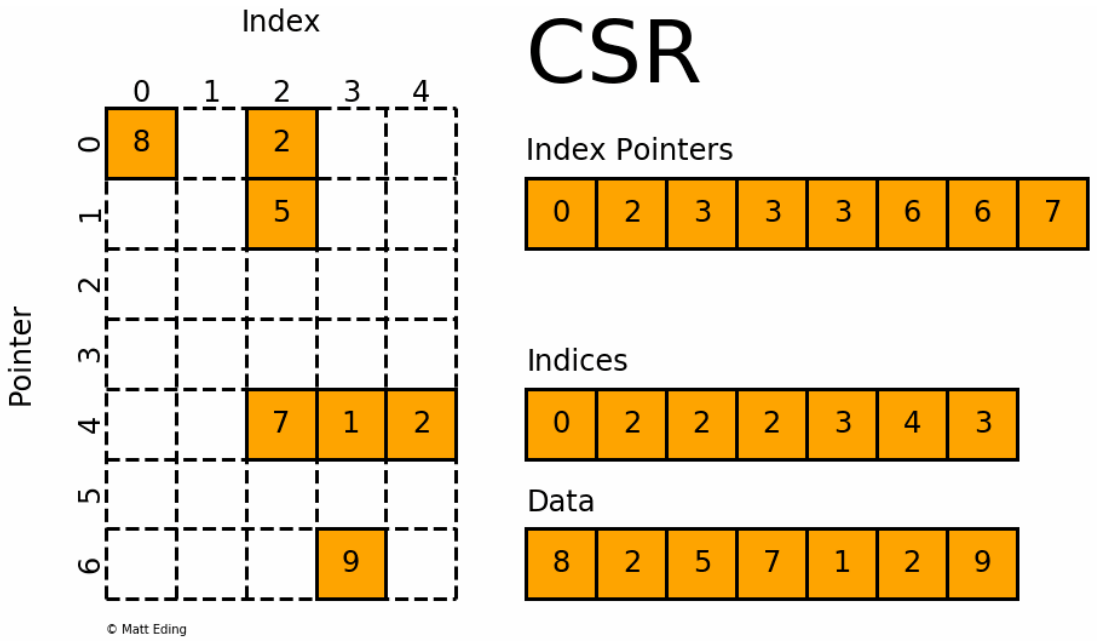


Image from https://matteding.github.io/2019/04/25/sparse-matrices

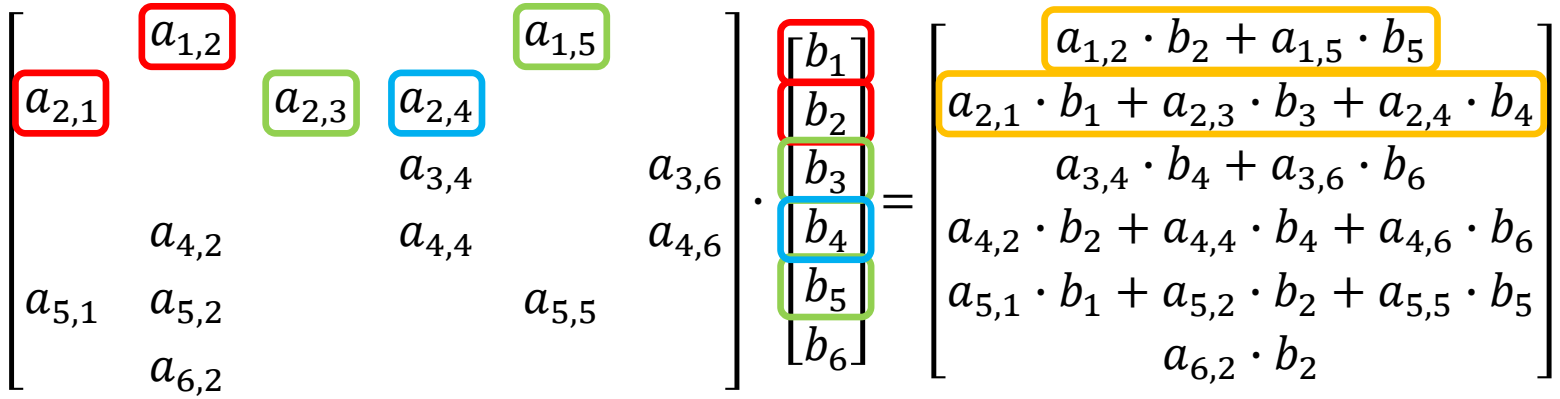
Dense Matrix-Vector multiplication

Multiply every element of each row with corresponding element of vector

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} & a_{1,6} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} & a_{2,6} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} & a_{3,6} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & a_{4,5} & a_{4,6} \\ a_{5,1} & a_{5,2} & a_{5,3} & a_{5,4} & a_{5,5} & a_{5,6} \\ a_{6,1} & a_{6,2} & a_{6,3} & a_{6,4} & a_{6,5} & a_{6,6} \end{bmatrix} \cdot \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{bmatrix} = \begin{bmatrix} a_{1,1} \cdot b_1 + a_{1,2} \cdot b_2 + a_{1,3} \cdot b_3 + a_{1,4} \cdot b_4 + a_{1,5} \cdot b_5 + a_{1,6} \cdot b_6 \\ a_{2,1} \cdot b_1 + a_{2,2} \cdot b_2 + a_{2,3} \cdot b_3 + a_{2,4} \cdot b_4 + a_{2,5} \cdot b_5 + a_{2,6} \cdot b_6 \\ a_{3,1} \cdot b_1 + a_{3,2} \cdot b_2 + a_{3,3} \cdot b_3 + a_{3,4} \cdot b_4 + a_{3,5} \cdot b_5 + a_{3,6} \cdot b_6 \\ a_{4,1} \cdot b_1 + a_{4,2} \cdot b_2 + a_{4,3} \cdot b_3 + a_{4,4} \cdot b_4 + a_{4,5} \cdot b_5 + a_{4,6} \cdot b_6 \\ a_{5,1} \cdot b_1 + a_{5,2} \cdot b_2 + a_{5,3} \cdot b_3 + a_{5,4} \cdot b_4 + a_{5,5} \cdot b_5 + a_{5,6} \cdot b_6 \\ a_{6,1} \cdot b_1 + a_{6,2} \cdot b_2 + a_{6,3} \cdot b_3 + a_{6,4} \cdot b_4 + a_{6,5} \cdot b_5 + a_{6,6} \cdot b_6 \end{bmatrix}$$

Sparse Matrix-Vector multiplication

Only non-zero elements of each row need to be multiplied with corresponding elements of vector



Sparse Matrix-Vector multiplication (COO)

```
struct SparseMatrixCOO {
  int    *Row;
  int    *Column;
  double *Data;
  int NNZ; // Non-zero elems.
  int N;   // Size of matrix.
};
```

All vectors are of length NNZ

```
void SpMV_COO(SparseMatrixCOO A, double *x, double *y) {
  for (int i = 0; i < A.N; i++) {
    y[i] = 0.0
  }

  for (int i = 0; i < A.NNZ; i++) {
    y[A.Row[i]] += A.Data[i] * x[A.Column[i]];
  }
}
```

Sparse Matrix-Vector multiplication (CSR)

```
struct SparseMatrixCSR {
  int    *IndexPointers;
  int    *Indices;
  double *Data;
  int NNZ; // Non-zero elems.
  int N;   // Size of matrix.
};
```

`IndexPointers` is of length NNZ+1

`Indices` and `Data` are of length `IndexPointers[NNZ]`

```
void SpMV_CSR(SparseMatrixCSR A, double *x, double *y) {
  for (int row = 0; row < A.NNZ; row++) {
    y[row] = 0.0;
    int start = A.IndexPointers[row];
    int end   = A.IndexPointers[row+1];
    for (int element = start; element < end; element++) {
      y[row] += A.Data[element] * x[A.Indices[element]];
    }
  }
}
```

Position in `IndexPointers` determines row

Take adjacent elements. Range [start:end) determines range in vector `Indices` for current row

Non-zero element in sparse matrix

Extract column of element from vector `Indices` to access element of vector x being multiplied

cuSPARSE

cuSPARSE

An implementation of BLAS functionality for spare matrices from NVidia for its GPUs

- Web site: https://developer.nvidia.com/gpu-accelerated-libraries
- Documentation: https://docs.nvidia.com/cuda/cusparse/index.html

Provides two sets of APIs:

- cuSPARSE
  - Common API for most uses, single GPU
- cuSPARSELt
  - Lightweight library dedicated to Sparse Matrix-matrix Multiply (SpMM) operations

cuSPARSE

The library provides the following functionalities:

- Operations between a sparse vector and a dense vector
  - sum, dot product, scatter, gather
- Operations between a dense matrix and a sparse vector
  - multiplication
- Operations between a sparse matrix and a dense vector
  - multiplication, triangular solver, tridiagonal solver, pentadiagonal solver
- Operations between a sparse matrix and a dense matrix
  - multiplication, triangular solver, tridiagonal solver, pentadiagonal solver
- Operations between a sparse matrix and a sparse matrix
  - sum, multiplication
- Operations between dense matrices with output a sparse matrix
  - multiplication
- Sparse matrix preconditioners
  - Incomplete Cholesky Factorization (level 0), Incomplete LU Factorization (level 0)
- Reordering and Conversion operations between different sparse matrix storage formats

Supported matrix formats

Dense matrices

Sparse matrices in the following formats

- COO: Coordinate format
- CSR: Compressed Sparse Row
- CSC: Compressed Sparse Column
- SELL: Sliced Ellpack
- BSR: Block Sparse Row
- BLOCKED-ELL: Blocked Ellpack

Not all operations are supported for all formats

- Check the documentation for the operations required in your program to decide on the format to use

Matrices can be either zero-base indexed or one-base indexed

Error handling

cuSPARSE

- Data type: cusparseStatus_t
- Call to cuBLAS routine returns a value of the above type
  - CUSPARSE_STATUS_SUCCESS : Call to routine completed successfully
  - Otherwise, routine completed with an error

cuSPARSE library context

Data type cusparseHandle_t provides a handle to the cuSPARSE library context

Function cusparseCreate() must be called to initialize a handle

- cusparseDestroy() to destroy the handle

The handle is explicitly passed to every subsequent library function call

```
cusparseStatus_t cusparseStat;
cusparseHandle_t handle;
…
cusparseStat = cusparseCreate(&handle);
if (cusparseStat != CUSPARSE_STATUS_SUCCESS) {
  printf("cuSPARSE initialization failed\n");
  return EXIT_FAILURE;
}
```

Matrix and vectors descriptors

Sparse matrices are represented by the data type cusparseSpMatDescr_t

Sparse vectors are represented by the data type cusparseSpVecDescr_t

Dense vectors are represented by the data type cusparseDnVecDescr_t

A matrix or vector must be created by the appropriate function in a supported format

Sparse matrix-dense vector multiplication

```
cusparseStatus_t cusparseSpMV(cusparseHandle_t              handle,
                              cusparseOperation_t           opA,
                              const void*                   alpha,
                              cusparseConstSpMatDescr_t     matA,
                              cusparseConstDnVecDescr_t     vecX,
                              const void*                   beta,
                              cusparseDnVecDescr_t          vecY,
                              cudaDataType                  computeType,
                              cusparseSpMVAlg_t             alg,
                              void*                         externalBuffer)
```

Performs the operation $vecY \leftarrow alpha \cdot op(matA) \cdot vecX + beta \cdot vecY$

- $matA$ is a sparse matrix
- $vecX$ and vecY are dense vectors
- $alpha$ and $beta$ are scalars

Parameters of cusparseSpMV()

Param.	Meaning
handle	Handle to the cuSPARSE library context
opA	Operation op(A)
alpha	α scalar used for multiplication of type computeType
matA	Sparse matrix A
vecX	Dense vector X
beta	β scalar used for multiplication of type computeType
vecY	Dense vector Y
computeType	Datatype in which the computation is executed
alg	Algorithm for the computation
bufferSize	Number of bytes of workspace needed by cusparseSpMV
externalBuffer	Pointer to a workspace buffer of at least bufferSize bytes

op(A)	Meaning
CUSPARSE_OPERATION_NON_TRANSPOSE	A
CUSPARSE_OPERATION_TRANSPOSE	A^T
CUSPARSE_OPERATION_CONJUGATE_TRANSPOSE	A^H

The function supports uniform ($matA$, $vecX$, $vecY$ have the same precision) and mixed precision computations

- Consult the documentation for supported combinations of precision among the participating matrix and vectors

cusparseSpMV() example

```
#include <cuda_runtime_api.h> // cudaMalloc, cudaMemcpy, etc.
#include <cusparse.h>         // cusparseSpMV
#include <stdio.h>            // printf
#include <stdlib.h>           // EXIT_FAILURE

int main(void) {
  // Definitions for the host
  int   A_num_rows = 4, A_num_cols = 4, A_nnz = 9;
  int   hA_rows[]    = { 0, 0, 0, 1, 2, 2, 2, 3, 3 };
  int   hA_columns[] = { 0, 2, 3, 1, 0, 2, 3, 1, 3 };
  float hA_values[]  = { 1.0f, 2.0f, 3.0f, 4.0f, 5.0f,
                         6.0f, 7.0f, 8.0f, 9.0f };
  float hX[]         = { 1.0f, 2.0f, 3.0f, 4.0f };
  float hY[]         = { 0.0f, 0.0f, 0.0f, 0.0f };
  float hY_result[]  = { 19.0f, 8.0f, 51.0f, 52.0f };
  float alpha        = 1.0f, beta = 0.0f;

  // Definitions for managing memory on the GPU
  int   *dA_rows, *dA_columns;
  float *dA_values, *dX, *dY;
```

```
  // Required for cuSPARSE API
  cusparseHandle_t     handle     = NULL;
  cusparseSpMatDescr_t matA;
  cusparseDnVecDescr_t vecX, vecY;
  void*                dBuffer    = NULL;
  size_t               bufferSize = 0;

  // Allocate arrays on the GPU
  cudaMalloc((void**) &dA_rows,    A_nnz * sizeof(int));
  cudaMalloc((void**) &dA_columns, A_nnz * sizeof(int));
  cudaMalloc((void**) &dA_values,  A_nnz * sizeof(float));
  cudaMalloc((void**) &dX,         A_num_cols * sizeof(float));
  cudaMalloc((void**) &dY,         A_num_rows * sizeof(float));

  // Copy arrays from host to devide
  cudaMemcpy(dA_rows, hA_rows, A_nnz * sizeof(int), cudaMemcpyHostToDevice);
  cudaMemcpy(dA_columns, hA_columns, A_nnz * sizeof(int), cudaMemcpyHostToDevice);
  cudaMemcpy(dA_values, hA_values, A_nnz * sizeof(float), cudaMemcpyHostToDevice);
  cudaMemcpy(dX, hX, A_num_cols * sizeof(float), cudaMemcpyHostToDevice);
  cudaMemcpy(dY, hY, A_num_rows * sizeof(float), cudaMemcpyHostToDevice);
```

cusparseSpMV() example

```
cusparseCreate(&handle);

// Create sparse matrix A in COO format
cusparseCreateCoo(&matA, A_num_rows, A_num_cols, A_nnz,
                  dA_rows, dA_columns, dA_values, CUSPARSE_INDEX_32I,
                  CUSPARSE_INDEX_BASE_ZERO, CUDA_R_32F);

// Create dense vector X
cusparseCreateDnVec(&vecX, A_num_cols, dX, CUDA_R_32F);
// Create dense vector y
cusparseCreateDnVec(&vecY, A_num_rows, dY, CUDA_R_32F);

// Allocate an external buffer if needed
cusparseSpMV_bufferSize(handle, CUSPARSE_OPERATION_NON_TRANSPOSE,
                        &alpha, matA, vecX, &beta, vecY, CUDA_R_32F,
                        CUSPARSE_SPMV_ALG_DEFAULT, &bufferSize);
cudaMalloc(&dBuffer, bufferSize);

// Execute SpMV
cusparseSpMV(handle, CUSPARSE_OPERATION_NON_TRANSPOSE,
                     &alpha, matA, vecX, &beta, vecY, CUDA_R_32F,
                     CUSPARSE_SPMV_ALG_DEFAULT, dBuffer);
```

```
  // Destroy matrix/vector descriptors
  cusparseDestroySpMat(matA);
  cusparseDestroyDnVec(vecX);
  cusparseDestroyDnVec(vecY);
  cusparseDestroy(handle);

  // Device memory deallocation
  cudaFree(dBuffer);
  cudaFree(dA_rows);
  cudaFree(dA_columns);
  cudaFree(dA_values);
  cudaFree(dX);
  cudaFree(dY);

  return EXIT_SUCCESS;
}
```

cuSOLVER

cuSOLVER

A collection of dense and sparse direct linear solvers andEigensolvers from NVidia for its GPUs

- Web site: https://developer.nvidia.com/gpu-accelerated-libraries
- Documentation: https://docs.nvidia.com/cuda/cusolver/index.html

Provides two sets of APIs:

- cuSOLVER
  - Common API for most uses, single GPU
- cuSOLVERMG
  - Single node, multi-GPU

cuSOLVERMp

- A related library for distributed-memory systems
  - Multiple nodes
  - Multi-GPU nodes

cuSOLVER

cusolverDN: Key LAPACK dense solvers

- Dense Cholesky, LU, SVD, QR
- Applications include: optimization, Computer Vision, CFD

cusolverSP

- Sparse direct solvers
- Symmetric & generalized symmetric eigensolvers
- Applications include: Newton's method, Chemical Kinetics

cusolverRF

- Sparse refactorization solver
- Applications include: Chemistry, ODEs, Circuit simulation

CUDA Graphs

CUDA Graphs

A way to define and batch GPU operations as a graph rather than a sequence of stream launches

- Part of the CUDA programming model
  - https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#cuda-graphs
  - https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__GRAPH.html

CUDA Graphs

Optimization of workloads with many dependent tasks

- A CUDA Graph represents a sequence of GPU operations as a graph of nodes
  - Call to kernels
  - Memory operations
  - Calls to functions in libraries that support CUDA Graphs

Execution is launched as a single unit

- Reduces overhead associated with individual kernel launches
- Enables GPUs to optimize execution at a higher level

CUDA Graphs

Key Features of CUDA Graphs

- Improved Efficiency
  - Launch a whole sequence of operations as a single graph, instead of launching individual kernels or memory operations
- Complex Task Representation
  - Representation of complex workflows involving multiple kernel launches, memory transfers, and stream synchronization as a single graph
- Reusability
  - Once a CUDA Graph has been constructed, it can be reused multiple times, potentially improving performance of repetitive tasks

CUDA Graphs

A CUDA Graph can be created by

- Either capturing within the program flow execution of kernels, memory operations and other function calls that execute on the GPU
- Or creating graphs directly within the program

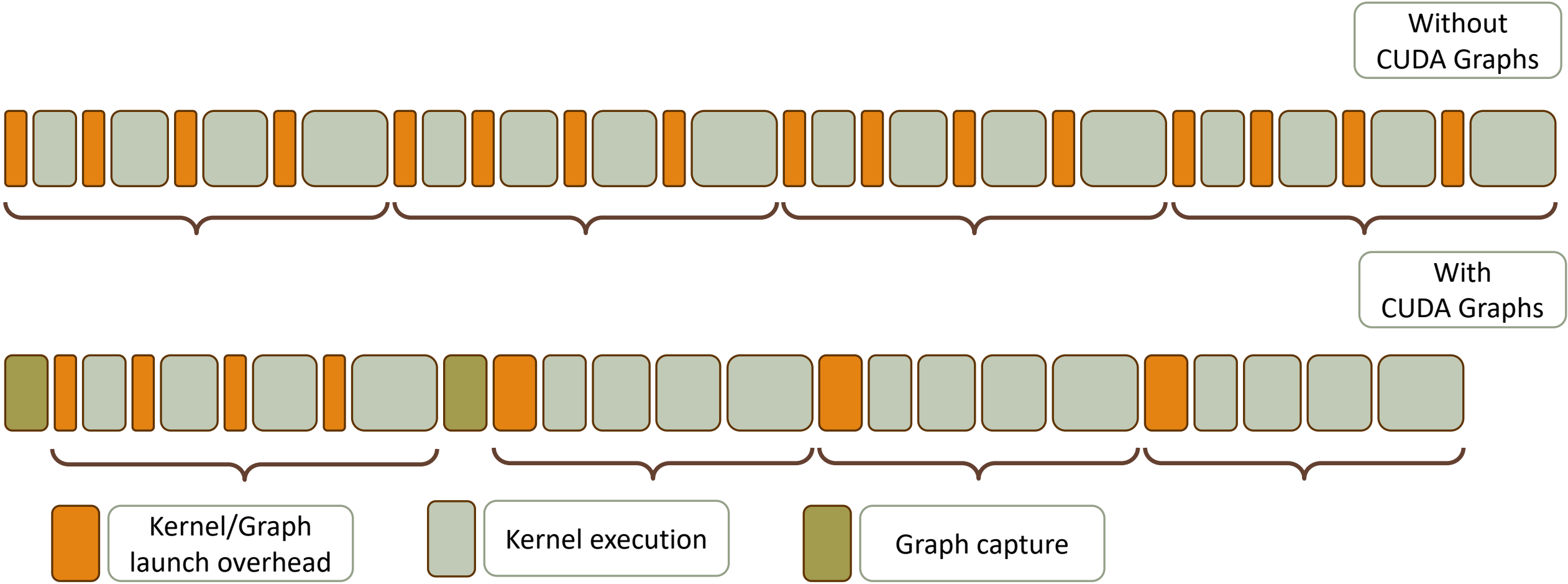
After creating a CUDA graph, it can be launched multiple times

Usage scenario

```
for (timestep = 0; timestep < T; timestep++) {
  …


  shortKernel1
  shortkernel2
  …
  shortKernelN









  …
}
```

```
graphCreated = false;
for (timestep = 0; timestep < T; timestep++) {
  …
  if (graphCreated == false) {
    startCapturingGraph()
    shortKernel1
    shortkernel2
    …
    shortKernelN
    stopCapturingGraph()
    instatiateGraph()
    graphCreated = true;
  } else {
    launchCapturedGraph()
  }
  …
}
```

Execution time



AmgX

AmgX

Provides distributed algebraic multigrid (AMG) and preconditioned iterative methods

- Web site: https://developer.nvidia.com/amgx
- Documentation: https://github.com/NVIDIA/AMGX/tree/main/doc/AMGX_Reference.pdf

The API presents an object-oriented framework for

- Describing a sparse linear system of equations
- A method for solving it

AmgX

MPI support
- Distributed-memory systems

OpenMP support
- In-node parallelism

Multi-GPU support

Well suited for implicit unstructured methods

Flexible configuration allows for nested solvers, smoothers, and preconditioners

Ruge-Steuben algebraic multigrid

Un-smoothed aggregation algebraic multigrid

Krylov methods
- PCG, GMRES, BiCGStab, and flexible variants

Smoothers
- Block-Jacobi, Gauss-Seidel, incomplete LU, Polynomial, dense LU

Scalar or coupled block systems

cuRAND

cuRAND

A GPU accelerated library for random number generation
- Web site: https://developer.nvidia.com/gpu-accelerated-libraries
- Documentation: https://docs.nvidia.com/cuda/curand/index.html

Different algorithms for pseudorandom andquasirandom number generation
- MRG32k3a
- MTGP Merseinne Twister
- XORWOW pseudo-random generation
- Sobol' quasi-random number generators, including support for scrambled and 64-bit RNG

Multiple RNG distribution options
- Uniform distribution
- Normal distribution
- Log-normal distribution
- Poisson distribution
- Single-precision or double-precision

Includes device level routines for RNG within user kernels

cuFFT

cuFFT

Fast Fourier Transform (FFT)functionality from NVidia for its GPUs

- Web site: https://developer.nvidia.com/gpu-accelerated-libraries
- Documentation: https://developer.nvidia.com/cufft

Provides two sets of APIs:

- cuFFT
  - Common API for most uses, single GPU
- cuFFTXt
  - Execution on multi-GPUs settings

Related libraries

- cuFFTMp
  - Multi-node support for FFTs in exascale problems
- cuFFTDx
  - For performing FFT calculations inside a CUDA kernel