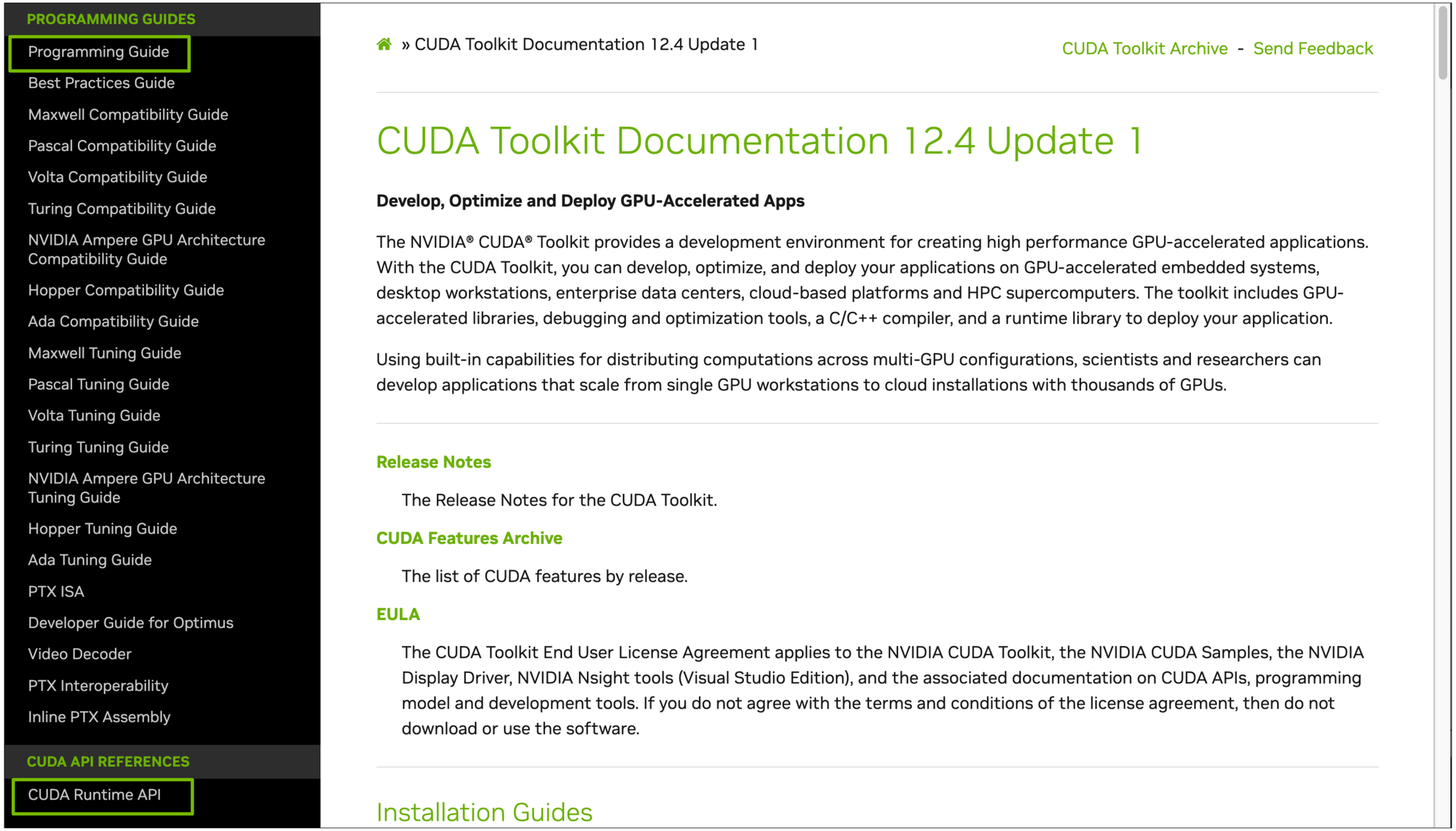
Hands-on (CUDA)

CUDA Toolkit Documentation

https://docs.nvidia.com/cuda/

Programming Guide

CUDA Runtime API



DAXPY

y[i] = a * x[i] + y[i]

```
//
//  daxpy: y[i] = a * x[i] + y[i]
//
void daxpy(int n, double a, double *x, double *y)
{
    int i;
    for (i = 0; i < n; i++) {
        y[i] += a*x[i];
    }
}
```

4 つのタスク

- Task 1: 配列を Unified Memory で確保する
- Task 2: GPU カーネルを実行できるようにする
- Task 3: Managed Memory 上のデータの移動を指示する
- Task 4: CPU と GPU 間で明示的に配列を確保、移動する

ディレクトリ構成、進め方

- タスクの進め方
  - ソースファイル (daxpy.cu) の中の /* TODO*/ の部分を修正して、正しく動作するようにする

reference	… リファレンス CPU コード
task1	… タスク 1
task1-solution	… タスク 1 の答え
task2	… タスク 2
task2-solution	… タスク 2 の答え
task3	… タスク 3
task3-solution	… タスク 3 の答え
task4	… タスク 4
task4-solution	… タスク 4 の答え

Task 0

教材コピー, ビルド, ジョブ投入

```
$ cp /gs/bs/GSIC/seminar/2026_Spring/gpu/hands-on.tar.gz .
$ tar xvzf hands-on.tar.gz
```

```
$ cd reference
$ module load cuda
$ make

$ qsub -g tgz-training -ar 6987 ./batch.sh
Your job **** ("batch.sh") has been submitted

$ qstat

$ ls batch.sh.*
batch.sh.e****  batch.sh.o****
```

```
#!/bin/sh
#$ -cwd
#$ -l node_q=1
#$ -l h_rt=0:05:00

module load cuda

./daxpy
```

batch.sh

Task 1

配列を Managed Memory で確保する

Managed Memory でメモリを確保する

- `cudaMallocManaged ( void** ptr, size_t size )`
  - `ptr` : 確保するメモリのポインタ
  - `size` : 確保するメモリサイズ (バイト)

メモリを開放する

- `cudaFree ( void** ptr )`
  - `ptr` : 開放するメモリのポインタ

Task 2

CUDA カーネルを実行できるようにする

CUDA カーネル内で使用できる定義済み変数

- `threadIdx.x` ： スレッドブロック内の、そのスレッドの ID
- `blockIdx.x` ： そのスレッドがいる、スレッドブロックの ID
- `blockDim.x` ： スレッドブロックの大きさ

カーネル実行条件

- `cuda_kernel<<<`ブロック数, スレッド数`>>>(...)`
- ブロック数 * スレッド数 ≧ 配列サイズ

GPU の実行完了を待つ

- `cudaDeviceSynchronize ( void )`

GPU は使われている？

プロファイラ (nsys) で確認する

```
#!/bin/sh
#$ -cwd

#$ -l node_q=1
#$ -l h_rt=00:05:00

module load cuda

nsys profile --stats=true -o report ./daxpy
nsys stats --report cuda_gpu_sum report.sqlite
```

batch.sh

```
...

Processing [report.sqlite] with [/apps/t4/rhel9/cuda/12.3.2/nsight-systems-2023.3.3/host-linux-x64/reports/cuda_gpu_sum.py]...

 ** CUDA GPU Summary (Kernels/MemOps) (cuda_gpu_sum):

 Time (%)  Total Time (ns)  Instances   Avg (ns)     Med (ns)    Min (ns)   Max (ns)   StdDev (ns)   Category                    Operation
 --------  ---------------  ---------  -----------  -----------  ---------  ---------  -----------  -----------  ----------------------------------
-------
     64.2        4,735,907          1  4,735,907.0  4,735,907.0  4,735,907  4,735,907          0.0  CUDA_KERNEL  cuda_daxpy(int, double, double *,
double *)
     25.9        1,914,489        622      3,078.0      2,399.0      2,143     30,080      2,513.1  MEMORY_OPER  [CUDA memcpy Unified Host-to-Device]
      9.9          731,468         96      7,619.5      2,911.5      2,271     39,647      9,733.5  MEMORY_OPER  [CUDA memcpy Unified Device-to-Host]
```

batch.sh.o*

Task 3

Managed Memory 上のデータの移動を指示する

Managed Memory 上のデータを、指定 GPU/CPU にプリフェッチする

- `cudaMemPrefetchAsync ( void** ptr, size_t size, cudaMemLocation location unsigned int flags )`
  - `ptr` : Unified Memory で確保したメモリのポインタ
  - `size` : プリフェッチサイズ (バイト)
  - `location` : プリフェッチ先
    - `cudaMemLocation::type` : `cudaMemLocationTypeDevice` で GPU へプリフェッチ
    - `cudaMemLocation::id` : プリフェッチ先の GPU 番号
    - `cudaMemLocation::type` : `cudaMemLocationTypeHost` で CPU へプリフェッチ
  - `flags` : 0 を指定 (今後の機能実装のためにリザーブされた引数)

CUDA カーネルの実行時間はどうなる？

プロファイラ (nsys) で確認する

```
#!/bin/sh
#$ -cwd

#$ -l node_q=1
#$ -l h_rt=00:05:00

module load cuda

nsys profile --stats=true -o report ./daxpy
nsys stats --report cuda_gpu_sum report.sqlite
```

batch.sh

```
...

Processing [report.sqlite] with [/apps/t4/rhel9/cuda/12.3.2/nsight-systems-2023.3.3/host-linux-x64/reports/cuda_gpu_sum.py]...

 ** CUDA GPU Summary (Kernels/MemOps) (cuda_gpu_sum):

 Time (%)  Total Time (ns)  Instances  Avg (ns)  Med (ns)  Min (ns)  Max (ns)  StdDev (ns)   Category                    Operation
 --------  ---------------  ---------  --------  --------  --------  --------  -----------  -----------  ----------------------------------------
     51.3          633,285         52  12,178.6   3,440.0     2,271    74,943     19,322.7  MEMORY_OPER  [CUDA memcpy Unified Device-to-Host]
     48.2          594,752          8  74,344.0  76,400.0    63,840    78,176      4,971.9  MEMORY_OPER  [CUDA memcpy Unified Host-to-Device]
      0.6            7,104          1   7,104.0   7,104.0     7,104     7,104          0.0  CUDA_KERNEL  cuda_daxpy(int, double, double *, double *)
```

batch.sh.o*

Task 4

CPU と GPU 間で明示的に配列を確保、移動する

デイバイスメモリを確保する

- `cudaMalloc ( void** ptr, size_t size )`
  - `ptr` : 確保するメモリのポインタ
  - `size` : 確保するメモリサイズ (バイト)

メモリを開放する

- `cudaFree ( void** ptr )`
  - `ptr` : 開放するメモリのポインタ

CPU と GPU 間でデータをコピーする

- `cudaMemcpy ( void* dst, void * src, size_t size, cudaMemcpyKind kind )`
  - `dst` : コピー先メモリのポインタ
  - `src` : コピー元メモリのポインタ
  - `size` : コピーサイズ (バイト)
  - `kind` : 移動方向
    - `cudaMemcpyHostToDevice` … CPU から GPU にコピー
    - `cudaMemcpyDeviceToHost` … GPU から CPU にコピー

データはどう転送される？

プロファイラ (nsys) で確認する

```
#!/bin/sh
#$ -cwd

#$ -l node_q=1
#$ -l h_rt=00:05:00

module load cuda

nsys profile --stats=true -o report ./daxpy
nsys stats --report cuda_gpu_sum report.sqlite
```

batch.sh

```
...

Processing [report.sqlite] with [/apps/t4/rhel9/cuda/12.3.2/nsight-systems-2023.3.3/host-linux-x64/reports/cuda_gpu_sum.py]...

 ** CUDA GPU Summary (Kernels/MemOps) (cuda_gpu_sum):

 Time (%)  Total Time (ns)  Instances  Avg (ns)   Med (ns)   Min (ns)  Max (ns)  StdDev (ns)   Category                    Operation
 --------  ---------------  ---------  ---------  ---------  --------  --------  -----------  -----------  ----------------------------------------
_
     64.5          627,164          2  313,582.0  313,582.0   310,334   316,830      4,593.4  MEMORY_OPER  [CUDA memcpy Host-to-Device]
     34.8          338,078          1  338,078.0  338,078.0   338,078   338,078          0.0  MEMORY_OPER  [CUDA memcpy Device-to-Host]
      0.7            7,041          1    7,041.0    7,041.0     7,041     7,041          0.0  CUDA_KERNEL  cuda_daxpy(int, double, double *, double
*)
```

batch.sh.o*