CUDA SDK — Basic Concepts

Siegfried Höfinger

ASC Research Center, TU Wien

October 20, 2025

→ https://tinyurl.com/cudafordummies/ii/l3/notes-l3.pdf

Outline

- Nowadays on github (curated, re-structured, toolkit-dependent)

→ https://github.com/nvidia/cuda-samples

- Nowadays on github (curated, re-structured, toolkit-dependent)
- Constantly growing collection of basic CUDA programming exercises (educational resource, not HPC-focussed)

→ https://github.com/nvidia/cuda-samples

Basic Conventions

CUDA SDK

- Nowadays on github (curated, re-structured, toolkit-dependent)
- Constantly growing collection of basic CUDA programming exercises (educational resource, not HPC-focussed)
- Showcases all GPU features and explains fundamental CUDA concepts

→ https://github.com/nvidia/cuda-samples

Basic Conventions

CUDA SDK

- Nowadays on github (curated, re-structured, toolkit-dependent)
- Constantly growing collection of basic CUDA programming exercises (educational resource, not HPC-focussed)
- Showcases all GPU features and explains fundamental CUDA concepts
- Open source and easily expandable with own projects

→ https://github.com/nvidia/cuda-samples

Basic Conventions

CUDA SDK

- Nowadays on github (curated, re-structured, toolkit-dependent)
- Constantly growing collection of basic CUDA programming exercises (educational resource, not HPC-focussed)
- Showcases all GPU features and explains fundamental CUDA concepts
- Open source and easily expandable with own projects
- Excellent place for beginners to start looking around

→ https://github.com/nvidia/cuda-samples

Basic Conventions

CUDA SDK

- Nowadays on github (curated, re-structured, toolkit-dependent)
- Constantly growing collection of basic CUDA programming exercises (educational resource, not HPC-focussed)
- Showcases all GPU features and explains fundamental CUDA concepts
- Open source and easily expandable with own projects
- Excellent place for beginners to start looking around
- Thematically organized into 9 major subject areas

→ https://github.com/nvidia/cuda-samples

Basic Conventions cont.

CUDA SDK cont.

```
cuda–zen sh@n3073–004:~$ git clone --branch v12.9 --single-branch https://github.com/NVIDIA/cuda-samples
cuda–zen sh@n3073–004:~$ cd cuda-samples/Samples
cuda–zen sh@n3073–004:Samples$ ls

0_Introduction  2_Concepts_and_Techniques  4_CUDA_Libraries   6_Performance  8_Platform_Specific
1_Utilities     3_CUDA_Features            5_Domain_Specific  7_libNVVM

cuda–zen sh@n3073–004:Samples$ cd 0_Introduction
cuda–zen sh@n3073–004:0_Introduction$ ls

CMakeLists.txt         mergeSort                      simpleHyperQ              simpleTemplates
README.md              simpleAWBarrier                simpleIPC                 simpleTexture
UnifiedMemoryStreams   simpleAssert                   simpleLayeredTexture      simpleTexture3D
asyncAPI               simpleAssert_nvrtc             simpleMPI                 simpleTextureDrv
clock                  simpleAtomicIntrinsics         simpleMultiCopy           simpleVoteIntrinsics
clock_nvrtc            simpleAtomicIntrinsics_nvrtc   simpleMultiGPU            simpleZeroCopy
cudaOpenMP             simpleAttributes               simpleOccupancy           systemWideAtomics
fp16ScalarProduct      simpleCUDA2GL                  simpleP2P                 template
matrixMul              simpleCallback                 simplePitchLinearTexture  vectorAdd
matrixMulDrv           simpleCooperativeGroups        simplePrintf              vectorAddDrv
matrixMulDynlinkJIT    simpleCubemapTexture           simpleStreams             vectorAddMMAP
matrixMul_nvrtc        simpleDrvRuntime               simpleSurfaceWrite        vectorAdd_nvrtc
```

→ https://github.com/nvidia/cuda-samples

Basic Conventions cont.

CUDA SDK cont.

```
cuda–zen sh@n3073–004:~$ git clone --branch v12.9 --single-branch https://github.com/NVIDIA/cuda-samples
cuda–zen sh@n3073–004:~$ cd cuda-samples/Samples
cuda–zen sh@n3073–004:Samples$ ls

0_Introduction  2_Concepts_and_Techniques  4_CUDA_Libraries   6_Performance  8_Platform_Specific
1_Utilities     3_CUDA_Features            5_Domain_Specific  7_libNVVM

cuda–zen sh@n3073–004:Samples$ cd 0_Introduction
cuda–zen sh@n3073–004:0_Introduction$ ls

CMakeLists.txt         mergeSort                     simpleHyperQ              simpleTemplates
README.md              simpleAWBarrier               simpleIPC                 simpleTexture
UnifiedMemoryStreams   simpleAssert                  simpleLayeredTexture      simpleTexture3D
asyncAPI               simpleAssert_nvrtc            simpleMPI                 simpleTextureDrv
clock                  simpleAtomicIntrinsics        simpleMultiCopy           simpleVoteIntrinsics
clock_nvrtc            simpleAtomicIntrinsics_nvrtc  simpleMultiGPU            simpleZeroCopy
cudaOpenMP             simpleAttributes              simpleOccupancy           systemWideAtomics
fp16ScalarProduct      simpleCUDA2GL                 simpleP2P                 template
matrixMul              simpleCallback                simplePitchLinearTexture  vectorAdd
matrixMulDrv           simpleCooperativeGroups       simplePrintf              vectorAddDrv
matrixMulDynlinkJIT    simpleCubemapTexture          simpleStreams             vectorAddMMAP
matrixMul_nvrtc        simpleDrvRuntime              simpleSurfaceWrite        vectorAdd_nvrtc
```

→ https://github.com/nvidia/cuda-samples

Basic Conventions cont.

CUDA SDK cont.

0_Introduction	CUDA samples for beginners that illustrate key concepts with using CUDA and CUDA runtime APIs
1_Utilities	Utility samples that demonstrate how to query device capabilities and measure GPU/CPU bandwidth
2_Concepts_and_Techniques	CUDA related concepts and common problem solving techniques
3_CUDA_Features	Samples that demonstrate CUDA Features (cooperative groups, dynamic parallelism, graphs etc)
4_CUDA_Libraries	Samples that illustrate how to use CUDA platform libraries, NPP, NVJPEG, NVGRAPH cuBLAS, cuFFT, cuSPARSE, cuSOLVER and cuRAND
5_Domain_Specific	Samples from specific domains (graphics, finance, image processing)
6_Performance	Samples that illustrate performance optimizations
7_libNVVM	Samples of interfacing with NVVM IR (intermediate representations)
8_Platform_Specific	Samples of using CUDA for TEGRA architectures

→ https://github.com/nvidia/cuda-samples

Basic Conventions cont.

CUDA SDK cont.

Consider for example function assert()

```
#include <stdio.h>
#include <assert.h>

int main()
{
    int i;

    for (i=0; i<10; i++) {
        assert(i < 5);
        printf("current i is %d \n", i);
    }

    return(0);
}
```

→ https://tinyurl.com/cudafordummies/ii/l3/smpl_assert.c

Basic Conventions cont.

CUDA SDK cont.

Consider for example function assert()

```
#include <stdio.h>
#include <assert.h>

int main()
{
    int i;

    for (i=0; i<10; i++) {
        assert(i < 5);
        printf("current i is %d \n", i);
    }

    return(0);
}
```

```
cuda-zen sh@n3073-004:~$  gcc ./smpl_assert.c
cuda-zen sh@n3073-004:~$  ./a.out
current i is 0
current i is 1
current i is 2
current i is 3
current i is 4
a.out: smpl_assert.c:20: main: Assertion `i < 5´ failed.
Aborted (core dumped)
```

→ https://tinyurl.com/cudafordummies/ii/l3/smpl_assert.c

Basic Conventions cont.

CUDA SDK cont.

Consider for example function assert()

```
#include <stdio.h>
#include <assert.h>

int main()
{
    int i;

    for (i=0; i<10; i++) {
        assert(i < 5);
        printf("current i is %d \n", i);
    }

    return(0);
}
```

Developer's checkpoints: If expression is TRUE assert() does nothing. If FALSE, abortion and error message.

```
cuda-zen sh@n3073-004:~$  gcc ./smpl_assert.c
cuda-zen sh@n3073-004:~$  ./a.out
current i is 0
current i is 1
current i is 2
current i is 3
current i is 4
a.out: smpl_assert.c:20: main: Assertion `i < 5' failed.
Aborted (core dumped)
```

→ https://tinyurl.com/cudafordummies/ii/l3/smpl_assert.c

Basic Conventions cont.

CUDA SDK cont.

There is also a CUDA SDK sample in 0_Introduction/simpleAssert/

Consider for example function assert()

```
#include <stdio.h>
#include <assert.h>

int main()
{
    int i;

    for (i=0; i<10; i++) {
        assert(i < 5);
        printf("current i is %d \n", i);
    }

    return(0);
}
```

Developer's checkpoints: If expression is TRUE assert() does nothing. If FALSE, abortion and error message.

```
cuda-zen sh@n3073-004:~$  gcc ./smpl_assert.c
cuda-zen sh@n3073-004:~$  ./a.out
current i is 0
current i is 1
current i is 2
current i is 3
current i is 4
a.out: smpl_assert.c:20: main: Assertion `i < 5' failed.
Aborted (core dumped)
```

→ https://tinyurl.com/cudafordummies/ii/l3/smpl_assert.c

Basic Conventions cont.

CUDA SDK cont. — 0_Introduction/simpleAssert

```
cuda-zen sh@n3073-004:~$ cd cuda-samples/Samples/0_Introduction/simpleAssert
cuda-zen sh@n3073-004:simpleAssert$ ls

CMakeLists.txt  README.md  simpleAssert.cu

cuda-zen sh@n3073-004:simpleAssert$ cd ../
cuda-zen sh@n3073-004:0_Introduction$ cp -r ./simpleAssert ./my_simpleAssert
cuda-zen sh@n3073-004:0_Introduction$ cd ./my_simpleAssert
cuda-zen sh@n3073-004:my_simpleAssert$ module purge
cuda-zen sh@n3073-004:my_simpleAssert$ module load gcc/12.2.0-gcc-8.5.0-7ih4ahb
cuda-zen sh@n3073-004:my_simpleAssert$ module load cuda/12.9.0-gcc-12.2.0-xznhx43
cuda-zen sh@n3073-004:my_simpleAssert$ module load cmake/3.29.1-gcc-12.2.0-yooqtj6
cuda-zen sh@n3073-004:my_simpleAssert$ mkdir build
cuda-zen sh@n3073-004:my_simpleAssert$ cd build
cuda-zen sh@n3073-004:build$ cmake ..
cuda-zen sh@n3073-004:build$ make
```

→ https://github.com/nvidia/cuda-samples

Basic Conventions cont.

CUDA SDK cont. — 0_Introduction/simpleAssert cont.

```
cuda-zen sh@n3073-004: build$ ./simpleAssert
simpleAssert starting...
OS_System_Type.release = 4.18.0-477.10.1.el8_8.x86_64
OS Info: <#1 SMP Tue May 16 07:35:04 EDT 2023>

GPU Device 0: "Ampere" with compute capability 8.0

Launch kernel to generate assertion failures

-- Begin assert output

simpleAssert.cu:63: void testKernel(int): block: [1,0,0], thread: [28,0,0] Assertion `gtid < N` failed.
simpleAssert.cu:63: void testKernel(int): block: [1,0,0], thread: [29,0,0] Assertion `gtid < N` failed.
simpleAssert.cu:63: void testKernel(int): block: [1,0,0], thread: [30,0,0] Assertion `gtid < N` failed.
simpleAssert.cu:63: void testKernel(int): block: [1,0,0], thread: [31,0,0] Assertion `gtid < N` failed.

-- End assert output

Device assert failed as expected, CUDA error message is: device-side assert triggered

simpleAssert completed, returned OK
```

→ https://github.com/nvidia/cuda-samples

Basic Conventions cont.

CUDA SDK cont. — 0_Introduction/simpleAssert cont.

simpleAssert.cu

```
__global__ void testKernel(int N)
{
    int gtid = blockIdx.x*blockDim.x + threadIdx.x;
    assert(gtid < N);
}
int main(int argc, char **argv)
{
    ...
    runTest(argc, argv);
    ...
}
void runTest(int argc, char **argv)
{
    int Nblocks = 2;
    int Nthreads = 32;
    cudaError_t error;
    ...
    findCudaDevice(argc, (const char **)argv);
    dim3 dimGrid(Nblocks);
    dim3 dimBlock(Nthreads);
    testKernel <<< dimGrid, dimBlock >>> (60);
    error = cudaDeviceSynchronize();
}
```

→ https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__TYPES.html

Basic Conventions cont.

CUDA SDK cont. — 0_Introduction/simpleAssert cont.

simpleAssert.cu

```
__global__ void testKernel(int N)
{
    int gtid = blockIdx.x*blockDim.x + threadIdx.x;
    assert(gtid < N);
}
int main(int argc, char **argv)
{
    ...
    runTest(argc, argv);
    ...
}
void runTest(int argc, char **argv)
{
    int Nblocks = 2;
    int Nthreads = 32;
    cudaError_t error;
    ...
    findCudaDevice(argc, (const char **)argv);
    dim3 dimGrid(Nblocks);
    dim3 dimBlock(Nthreads);
    testKernel <<< dimGrid, dimBlock >>> (60);
    error = cudaDeviceSynchronize();
}
```

2 threadblocks with 32 threads

→ https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__TYPES.html

Basic Conventions cont.

CUDA SDK cont. — 0_Introduction/simpleAssert cont.

simpleAssert.cu

```
__global__ void testKernel(int N)
{
    int gtid = blockIdx.x*blockDim.x + threadIdx.x;
    assert(gtid < N);
}
int main(int argc, char **argv)
{
    ...
    runTest(argc, argv);
    ...
}
void runTest(int argc, char **argv)
{
    int Nblocks = 2;
    int Nthreads = 32;
    cudaError_t error;
    ...
    findCudaDevice(argc, (const char **)argv);
    dim3 dimGrid(Nblocks);
    dim3 dimBlock(Nthreads);
    testKernel <<< dimGrid, dimBlock >>> (60);
    error = cudaDeviceSynchronize();
}
```

New type (enum) for error variable

2 threadblocks with 32 threads

→ https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__TYPES.html

Basic Conventions cont.

CUDA SDK cont. — 0_Introduction/simpleAssert cont.

simpleAssert.cu

```
__global__ void testKernel(int N)
{
    int gtid = blockIdx.x*blockDim.x + threadIdx.x;
    assert(gtid < N);
}
int main(int argc, char **argv)
{
    ...
    runTest(argc, argv);
    ...
}
void runTest(int argc, char **argv)
{
    int Nblocks = 2;
    int Nthreads = 32;
    cudaError_t error;
    ...
    findCudaDevice(argc, (const char **)argv);
    dim3 dimGrid(Nblocks);
    dim3 dimBlock(Nthreads);
    testKernel <<< dimGrid, dimBlock >>> (60);
    error = cudaDeviceSynchronize();
}
```

New type (enum) for error variable

Generic function to identify GPU

2 threadblocks with 32 threads

→ https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__TYPES.html

Basic Conventions cont.

CUDA SDK cont. — 0_Introduction/simpleAssert cont.

simpleAssert.cu

```
__global__ void testKernel(int N)
{
    int gtid = blockIdx.x*blockDim.x + threadIdx.x;
    assert(gtid < N);
}
int main(int argc, char **argv)
{
    ...
    runTest(argc, argv);
    ...
}
void runTest(int argc, char **argv)
{
    int Nblocks = 2;
    int Nthreads = 32;
    cudaError_t error;
    ...
    findCudaDevice(argc, (const char **)argv);
    dim3 dimGrid(Nblocks);
    dim3 dimBlock(Nthreads);
    testKernel <<< dimGrid, dimBlock >>> (60);
    error = cudaDeviceSynchronize();
}
```

New type (enum) for error variable

Generic function to identify GPU

2 threadblocks with 32 threads

C++ style of initializing variables dimGrid and dimBlock

→ https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__TYPES.html

Basic Conventions cont.

CUDA SDK cont. — 0_Introduction/simpleAssert cont.

simpleAssert.cu

```
__global__ void testKernel(int N)
{
    int gtid = blockIdx.x*blockDim.x + threadIdx.x;
    assert(gtid < N);
}
int main(int argc, char **argv)
{
    ...
    runTest(argc, argv);
    ...
}
void runTest(int argc, char **argv)
{
    int Nblocks = 2;
    int Nthreads = 32;
    cudaError_t error;
    ...
    findCudaDevice(argc, (const char **)argv);
    dim3 dimGrid(Nblocks);
    dim3 dimBlock(Nthreads);
    testKernel <<< dimGrid, dimBlock >>> (60);
    error = cudaDeviceSynchronize();
}
```

New type (enum) for error variable

Generic function to identify GPU

2 threadblocks with 32 threads

C++ style of initializing variables dimGrid and dimBlock

kernel launch

→ https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__TYPES.html

Basic Conventions cont.

CUDA SDK cont. — 0_Introduction/simpleAssert cont.

simpleAssert.cu

```
__global__ void testKernel(int N)
{
    int gtid = blockIdx.x*blockDim.x + threadIdx.x;
    assert(gtid < N);
}
int main(int argc, char **argv)
{
    ...
    runTest(argc, argv);
    ...
}
void runTest(int argc, char **argv)
{
    int Nblocks = 2;
    int Nthreads = 32;
    cudaError_t error;
    ...
    findCudaDevice(argc, (const char **)argv);
    dim3 dimGrid(Nblocks);
    dim3 dimBlock(Nthreads);
    testKernel <<< dimGrid, dimBlock >>> (60);
    error = cudaDeviceSynchronize();
}
```

Check is whether gtid is < 60

2 threadblocks with 32 threads

New type (enum) for error variable

Generic function to identify GPU

C++ style of initializing variables dimGrid and dimBlock

kernel launch

→ https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__TYPES.html

Basic Conventions cont.

CUDA SDK cont. — 0_Introduction/simpleAssert cont.

simpleAssert.cu

```
__global__ void testKernel(int N)
{
    int gtid = blockIdx.x*blockDim.x + threadIdx.x;
    assert(gtid < N);
}
int main(int argc, char **argv)
{
    ...
    runTest(argc, argv);
    ...
}
void runTest(int argc, char **argv)
{
    int Nblocks = 2;
    int Nthreads = 32;
    cudaError_t error;
    ...
    findCudaDevice(argc, (const char **)argv);
    dim3 dimGrid(Nblocks);
    dim3 dimBlock(Nthreads);
    testKernel <<< dimGrid, dimBlock >>> (60);
    error = cudaDeviceSynchronize();
}
```

Device behaviour identical to host; lots of useful information !

New type (enum) for error variable

Generic function to identify GPU

Check is whether gtid is < 60

2 threadblocks with 32 threads

C++ style of initializing variables dimGrid and dimBlock

kernel launch

→ https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__TYPES.html

simpleAssert.cu / error handling

```
void runTest(int argc, char **argv)
{
    ...
    printf("Launch kernel to generate assertion failures\n");
    testKernel <<< dimGrid, dimBlock >>> (60);

    //Synchronize (flushes assert output)
    printf("\n- Begin assert output\n\n");
    error = cudaDeviceSynchronize();
    printf("\n- End assert output\n\n");

    //Check for errors
    if (error == cudaErrorAssert) {
        printf("Device assert failed as expected, "
               "CUDA error message is: %s\n\n",
               cudaGetErrorString(error));
    }
    testResult = error == cudaErrorAssert;
}
```

→ https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__TYPES.html

simpleAssert.cu / error handling

```
void runTest(int argc, char **argv)
{
    ...
    printf("Launch kernel to generate assertion failures\n");
    testKernel <<< dimGrid, dimBlock >>> (60);

    //Synchronize (flushes assert output)
    printf("\n- Begin assert output\n\n");
    error = cudaDeviceSynchronize();
    printf("\n- End assert output\n\n");

    //Check for errors
    if (error == cudaErrorAssert) {
        printf("Device assert failed as expected, "
               "CUDA error message is: %s\n\n",
               cudaGetErrorString(error));
    }
    testResult = error == cudaErrorAssert;
}
```

Pick up return value from CUDA call

→ https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__TYPES.html

simpleAssert.cu / error handling

```
void runTest(int argc, char **argv)
{
    ...
    printf("Launch kernel to generate assertion failures\n");
    testKernel <<< dimGrid, dimBlock >>> (60);

    //Synchronize (flushes assert output)
    printf("\n- Begin assert output\n\n");
    error = cudaDeviceSynchronize();
    printf("\n- End assert output\n\n");

    //Check for errors
    if (error == cudaErrorAssert) {
        printf("Device assert failed as expected, "
               "CUDA error message is: %s\n\n",
               cudaGetErrorString(error));
    }
    testResult = error == cudaErrorAssert;
}
```

Pick up return value from CUDA call

Get specific info from returned error

→ https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__TYPES.html

simpleAssert.cu / error handling

```
void runTest(int argc, char **argv)
{
    ...
    printf("Launch kernel to generate assertion failures\n");
    testKernel <<< dimGrid, dimBlock >>> (60);

    //Synchronize (flushes assert output)
    printf("\n- Begin assert output\n\n");
    error = cudaDeviceSynchronize();
    printf("\n- End assert output\n\n");

    //Check for errors
    if (error == cudaErrorAssert) {
        printf("Device assert failed as expected, "
               "CUDA error message is: %s\n\n",
               cudaGetErrorString(error));
    }
    testResult = error == cudaErrorAssert;
}
```

Pick up return value from CUDA call

Get specific info from returned error

Set global variable to true/false

→ https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__TYPES.html

- assert() is a very simple and convenient way to do low-level debugging of kernel code

→ https://github.com/nvidia/cuda-samples

- assert() is a very simple and convenient way to do low-level debugging of kernel code
- Returns very detailed information, threadIdx, blockIdx, line number, function name

→ https://github.com/nvidia/cuda-samples

Basic Conventions cont.

CUDA SDK cont. — 0_Introduction/simpleAssert cont.

- assert() is a very simple and convenient way to do low-level debugging of kernel code
- Returns very detailed information, threadIdx, blockIdx, line number, function name
- With printf() — surprisingly — we do also get output written from kernel code sections, however only at full block level terminating correctly

→ https://github.com/nvidia/cuda-samples

- assert() is a very simple and convenient way to do low-level debugging of kernel code
- Returns very detailed information, threadIdx, blockIdx, line number, function name
- With printf() — surprisingly — we do also get output written from kernel code sections, however only at full block level terminating correctly
- For example add another line after assert(gtid $<$ N);
  printf("*** message from thread %d ***\n", gtid);

→ https://github.com/nvidia/cuda-samples

Basic Conventions cont.

CUDA SDK cont. — 0_Introduction/simpleAssert cont.

```
cuda-zen sh@n3073-004: build$ ./simpleAssert

simpleAssert starting...
OS_System_Type.release = 4.18.0-477.10.1.el8_8.x86_64
OS Info: <#1 SMP Tue May 16 07:35:04 EDT 2023>

GPU Device 0: "Ampere" with compute capability 8.0

Launch kernel to generate assertion failures

-- Begin assert output

*** message from thread 0 ***
*** message from thread 1 ***
*** message from thread 2 ***
......
*** message from thread 29 ***
*** message from thread 30 ***
*** message from thread 31 ***
simpleAssert.cu:63: void testKernel(int): block: [1,0,0], thread: [28,0,0] Assertion `gtid < N` failed.
simpleAssert.cu:63: void testKernel(int): block: [1,0,0], thread: [29,0,0] Assertion `gtid < N` failed.
simpleAssert.cu:63: void testKernel(int): block: [1,0,0], thread: [30,0,0] Assertion `gtid < N` failed.
simpleAssert.cu:63: void testKernel(int): block: [1,0,0], thread: [31,0,0] Assertion `gtid < N` failed.

-- End assert output
```

→ https://github.com/nvidia/cuda-samples

0_Introduction/simplePrintf

CUDA SDK cont.

- There is also another simple CUDA example demonstrating regular operation of printf() in kernel code sections running on the device
- Compute capability must be at least 2.0
- Otherwise an alternative cuPrintf() can be used
- This example is also a good exercise to recall basic builtin variables of the kernel code section, e.g. threadIdx, blockDim etc.

→ https://github.com/nvidia/cuda-samples

0_Introduction/simplePrintf cont.

CUDA SDK cont.

```
cuda-zen sh@n3073-004: 0_Introduction$   cp -r ./simplePrintf ./my_simplePrintf
cuda-zen sh@n3073-004: 0_Introduction$   cd ./my_simplePrintf
cuda-zen sh@n3073-004: my_simplePrintf$   mkdir build
cuda-zen sh@n3073-004: my_simplePrintf$   cd build
cuda-zen sh@n3073-004: build$   cmake ..
cuda-zen sh@n3073-004: build$   make
cuda-zen sh@n3073-004: build$   ./simplePrintf
GPU Device 0: "Ampere" with compute capability 8.0

Device 0: "NVIDIA A100-PCIE-40GB" with Compute 8.0 capability
printf() is called. Output:

[1, 0]:         Value is:10             [0, 0]:         Value is:10
[1, 1]:         Value is:10             [0, 1]:         Value is:10
[1, 2]:         Value is:10             [0, 2]:         Value is:10
[1, 3]:         Value is:10             [0, 3]:         Value is:10
[1, 4]:         Value is:10             [0, 4]:         Value is:10
[1, 5]:         Value is:10             [0, 5]:         Value is:10
[1, 6]:         Value is:10             [0, 6]:         Value is:10
[1, 7]:         Value is:10             [0, 7]:         Value is:10
[2, 0]:         Value is:10             [3, 0]:         Value is:10
[2, 1]:         Value is:10             [3, 1]:         Value is:10
[2, 2]:         Value is:10             [3, 2]:         Value is:10
[2, 3]:         Value is:10             [3, 3]:         Value is:10
[2, 4]:         Value is:10             [3, 4]:         Value is:10
[2, 5]:         Value is:10             [3, 5]:         Value is:10
[2, 6]:         Value is:10             [3, 6]:         Value is:10
[2, 7]:         Value is:10             [3, 7]:         Value is:10
```

0_Introduction/simplePrintf cont.

CUDA SDK cont.

```
cuda-zen sh@n3073-004: 0_Introduction$   cp -r ./simplePrintf ./my_simplePrintf
cuda-zen sh@n3073-004: 0_Introduction$   cd ./my_simplePrintf
cuda-zen sh@n3073-004: my_simplePrintf$   mkdir build
cuda-zen sh@n3073-004: my_simplePrintf$   cd build
cuda-zen sh@n3073-004: build$   cmake ..
cuda-zen sh@n3073-004: build$   make
cuda-zen sh@n3073-004: build$   ./simplePrintf
GPU Device 0: "Ampere" with compute capability 8.0

Device 0: "NVIDIA A100-PCIE-40GB" with Compute 8.0 capability
printf() is called. Output:

[1, 0]:         Value is:10             [0, 0]:         Value is:10
[1, 1]:         Value is:10             [0, 1]:         Value is:10
[1, 2]:         Value is:10             [0, 2]:         Value is:10
[1, 3]:         Value is:10             [0, 3]:         Value is:10
[1, 4]:         Value is:10             [0, 4]:         Value is:10
[1, 5]:         Value is:10             [0, 5]:         Value is:10
[1, 6]:         Value is:10             [0, 6]:         Value is:10
[1, 7]:         Value is:10             [0, 7]:         Value is:10
[2, 0]:         Value is:10             [3, 0]:         Value is:10
[2, 1]:         Value is:10             [3, 1]:         Value is:10
[2, 2]:         Value is:10             [3, 2]:         Value is:10
[2, 3]:         Value is:10             [3, 3]:         Value is:10
[2, 4]:         Value is:10             [3, 4]:         Value is:10
[2, 5]:         Value is:10             [3, 5]:         Value is:10
[2, 6]:         Value is:10             [3, 6]:         Value is:10
[2, 7]:         Value is:10             [3, 7]:         Value is:10
```

4 x blocks with indices from 0-7

0_Introduction/simplePrintf cont.

CUDA SDK cont.

```
cuda-zen sh@n3073-004: 0_Introduction$   cp -r ./simplePrintf ./my_simplePrintf
cuda-zen sh@n3073-004: 0_Introduction$   cd ./my_simplePrintf
cuda-zen sh@n3073-004: my_simplePrintf$   mkdir build
cuda-zen sh@n3073-004: my_simplePrintf$   cd build
cuda-zen sh@n3073-004: build$   cmake ..
cuda-zen sh@n3073-004: build$   make
cuda-zen sh@n3073-004: build$   ./simplePrintf
GPU Device 0: "Ampere" with compute capability 8.0

Device 0: "NVIDIA A100-PCIE-40GB" with Compute 8.0 capability
printf() is called. Output:

[1, 0]:         Value is:10             [0, 0]:         Value is:10
[1, 1]:         Value is:10             [0, 1]:         Value is:10
[1, 2]:         Value is:10             [0, 2]:         Value is:10
[1, 3]:         Value is:10             [0, 3]:         Value is:10
[1, 4]:         Value is:10             [0, 4]:         Value is:10
[1, 5]:         Value is:10             [0, 5]:         Value is:10
[1, 6]:         Value is:10             [0, 6]:         Value is:10
[1, 7]:         Value is:10             [0, 7]:         Value is:10
[2, 0]:         Value is:10             [3, 0]:         Value is:10
[2, 1]:         Value is:10             [3, 1]:         Value is:10
[2, 2]:         Value is:10             [3, 2]:         Value is:10
[2, 3]:         Value is:10             [3, 3]:         Value is:10
[2, 4]:         Value is:10             [3, 4]:         Value is:10
[2, 5]:         Value is:10             [3, 5]:         Value is:10
[2, 6]:         Value is:10             [3, 6]:         Value is:10
[2, 7]:         Value is:10             [3, 7]:         Value is:10
```

4 x blocks with indices from 0-7

stochastic order of blocks (1st index)

0_Introduction/simplePrintf cont.

CUDA SDK cont.

simplePrintf.cu

```
__global__ void testKernel(int val)
{
    int gtid = blockIdx.x*blockDim.x + threadIdx.x;
    printf("[%d, %d]:\t\tValue is:%d\n",
           blockIdx.y*gridDim.x+blockIdx.x,
           threadIdx.z*blockDim.x*blockDim.y+threadIdx.y*blockDim.x+threadIdx.x,
           val);
}

int main(int argc, char **argv)
{
    ...
    dim3 dimGrid(2, 2);
    dim3 dimBlock(2, 2, 2);
    testKernel <<< dimGrid, dimBlock >>> (10);
    error = cudaDeviceSynchronize();
    ...
}
```

→ https://github.com/nvidia/cuda-samples

0_Introduction/simplePrintf cont.

CUDA SDK cont.

simplePrintf.cu

```
__global__ void testKernel(int val)
{
    int gtid = blockIdx.x*blockDim.x + threadIdx.x;
    printf("[%d, %d]:\t\tValue is:%d\n",
            blockIdx.y*gridDim.x+blockIdx.x,
            threadIdx.z*blockDim.x*blockDim.y+threadIdx.y*blockDim.x+threadIdx.x,
            val);
}

int main(int argc, char **argv)
{
    ...
    dim3 dimGrid(2, 2);
    dim3 dimBlock(2, 2, 2);
    testKernel <<< dimGrid, dimBlock >>> (10);
    error = cudaDeviceSynchronize();
    ...
}
```

4 threadblocks (2D-grid) with 8 threads (3D-block)

→ https://github.com/nvidia/cuda-samples

0_Introduction/simplePrintf cont.

CUDA SDK cont.

simplePrintf.cu

```
__global__ void testKernel(int val)
{
    int gtid = blockIdx.x*blockDim.x + threadIdx.x;
    printf("[%d, %d]:\t\tValue is:%d\n",
            blockIdx.y*gridDim.x+blockIdx.x,
            threadIdx.z*blockDim.x*blockDim.y+threadIdx.y*blockDim.x+threadIdx.x,
            val);
}

int main(int argc, char **argv)
{
    ...
    dim3 dimGrid(2, 2);
    dim3 dimBlock(2, 2, 2);
    testKernel <<< dimGrid, dimBlock >>> (10);
    error = cudaDeviceSynchronize();
    ...
}
```

Linearizes 2D-grid

4 threadblocks (2D-grid) with 8 threads (3D-block)

→ https://github.com/nvidia/cuda-samples

0_Introduction/simplePrintf cont.

CUDA SDK cont.

simplePrintf.cu

```
__global__ void testKernel(int val)
{
    int gtid = blockIdx.x*blockDim.x + threadIdx.x;
    printf("[%d, %d]:\t\tValue is:%d\n",
           blockIdx.y*gridDim.x+blockIdx.x,
           threadIdx.z*blockDim.x*blockDim.y+threadIdx.y*blockDim.x+threadIdx.x,
           val);
}

int main(int argc, char **argv)
{
    ...
    dim3 dimGrid(2, 2);
    dim3 dimBlock(2, 2, 2);
    testKernel <<< dimGrid, dimBlock >>> (10);
    error = cudaDeviceSynchronize();
    ...
}
```

Linearizes 2D-grid

Again, serialization of 3D-threadblocks

4 threadblocks (2D-grid) with 8 threads (3D-block)

→ https://github.com/nvidia/cuda-samples

- Good to see that printf() can be used also in kernel code
- Out of order execution of individual threadblocks in the blockgrid
- Only when the entire threadblock terminates correctly, printf() output will actually show up
- Important for the developmental stage, probably too expensive for production-ready runs

→ https://github.com/nvidia/cuda-samples

1_Utilities
CUDA SDK cont.

- 1_Utilities contains several CUDA examples that may also be regarded as simple tools to characterize the GPU hardware at hand

→ https://github.com/nvidia/cuda-samples

1_Utilities
CUDA SDK cont.

- 1_Utilities contains several CUDA examples that may also be regarded as simple tools to characterize the GPU hardware at hand
- deviceQuery has already been presented as a useful standard tool to list all device properties

→ https://github.com/nvidia/cuda-samples

1_Utilities
CUDA SDK cont.

- 1_Utilities contains several CUDA examples that may also be regarded as simple tools to characterize the GPU hardware at hand
- deviceQuery has already been presented as a useful standard tool to list all device properties
- A recurring issue with CUDA is bandwidth of data transfer — especially because there are so many different variants

→ https://github.com/nvidia/cuda-samples

1_Utilities
CUDA SDK cont.

- 1_Utilities contains several CUDA examples that may also be regarded as simple tools to characterize the GPU hardware at hand
- deviceQuery has already been presented as a useful standard tool to list all device properties
- A recurring issue with CUDA is bandwidth of data transfer — especially because there are so many different variants
- The bandwidthTest example is nowadays provided as a demo — it helps to get a quick overview of what bandwidth we can expect on the current device

→ https://github.com/nvidia/cuda-samples

1_Utilities
CUDA SDK cont.

- 1_Utilities contains several CUDA examples that may also be regarded as simple tools to characterize the GPU hardware at hand
- deviceQuery has already been presented as a useful standard tool to list all device properties
- A recurring issue with CUDA is bandwidth of data transfer — especially because there are so many different variants
- The bandwidthTest example is nowadays provided as a demo — it helps to get a quick overview of what bandwidth we can expect on the current device
- There are also several CLI args that may provide guidance for size/type dependence

→ https://github.com/nvidia/cuda-samples

1_Utilities cont.

CUDA SDK cont.

```
cuda-zen sh@n3073-004:1_Utilities$ which nvcc
cuda-zen sh@n3073-004:1_Utilities$ /gpfs/opt/sw/cuda-zen/spack-0.19.0/opt/spack/linux-almalinux8-zen/gcc-12.2.0/cuda-12.9.0-xznhx43d6xdwudiuv7xcutpx6ympzybi/extras/demo_suite/bandwidthTest


[CUDA Bandwidth Test] - Starting...
Running on...

 Device 0: NVIDIA A100-PCIE-40GB
 Quick Mode

 Host to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     26114.0

 Device to Host Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     25440.6

 Device to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     1205414.9
```

→ https://github.com/nvidia/cuda-samples

1_Utilities cont.

CUDA SDK cont.

```
cuda-zen sh@n3073-004: 1_Utilities$ which nvcc
cuda-zen sh@n3073-004: 1_Utilities$ /gpfs/opt/sw/cuda-zen/spack-0.19.0/opt/spack/linux-almalinux8-zen/gcc-12.2.0/cuda-12.9.0-xznhx43d6xdwudiuv7xcutpx6ympzybi/extras/demo_suite/bandwidthTest


[CUDA Bandwidth Test] - Starting...
Running on...

 Device 0: NVIDIA A100-PCIE-40GB
 Quick Mode

 Host to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     26114.0

 Device to Host Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     25440.6

 Device to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     1205414.9
```

32MB (pinned) with ≈25 GB/s

→ https://github.com/nvidia/cuda-samples

1_Utilities cont.

CUDA SDK cont.

```
cuda-zen sh@n3073-004:1_Utilities$ /gpfs/opt/sw/cuda-zen/spack-0.19.0/opt/spack/linux-almalinux8-zen/gcc-12.2.0/cuda-12.9.0-xznhx43d6xdwudiuv7xcutpx6ympzybi/extras/demo_suite/bandwidthTest --help
cuda-zen sh@n3073-004:1_Utilities$ /gpfs/opt/sw/cuda-zen/spack-0.19.0/opt/spack/linux-almalinux8-zen/gcc-12.2.0/cuda-12.9.0-xznhx43d6xdwudiuv7xcutpx6ympzybi/extras/demo_suite/bandwidthTest --mode=quick --memory=pageable --htod
[CUDA Bandwidth Test] - Starting...
Running on...

 Device 0: NVIDIA A100-PCIE-40GB
 Quick Mode

 Host to Device Bandwidth, 1 Device(s)
 PAGEABLE Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     16050.9

cuda-zen sh@n3073-004:1_Utilities$ /gpfs/opt/sw/cuda-zen/spack-0.19.0/opt/spack/linux-almalinux8-zen/gcc-12.2.0/cuda-12.9.0-xznhx43d6xdwudiuv7xcutpx6ympzybi/extras/demo_suite/bandwidthTest --mode=quick --memory=pinned --htod
[CUDA Bandwidth Test] - Starting...
Running on...

 Device 0: NVIDIA A100-PCIE-40GB
 Quick Mode

 Host to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     26127.6
```

→ https://github.com/nvidia/cuda-samples

1_Utilities cont.

CUDA SDK cont.

```
cuda-zen sh@n3073-004: 1_Utilities$ /gpfs/opt/sw/cuda-zen/spack-0.19.0/opt/spack/linux-almalinux8-zen/gcc-12.2.0/cuda-12.9.0-xznhx43d6xdwudiuv7xcutpx6ympzybi/extras/demo_suite/bandwidthTest --help
cuda-zen sh@n3073-004: 1_Utilities$ /gpfs/opt/sw/cuda-zen/spack-0.19.0/opt/spack/linux-almalinux8-zen/gcc-12.2.0/cuda-12.9.0-xznhx43d6xdwudiuv7xcutpx6ympzybi/extras/demo_suite/bandwidthTest --mode=quick --memory=pageable --htod
[CUDA Bandwidth Test] - Starting...
Running on...

 Device 0: NVIDIA A100-PCIE-40GB
 Quick Mode

 Host to Device Bandwidth, 1 Device(s)
 PAGEABLE Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     16050.9

cuda-zen sh@n3073-004: 1_Utilities$ /gpfs/opt/sw/cuda-zen/spack-0.19.0/opt/spack/linux-almalinux8-zen/gcc-12.2.0/cuda-12.9.0-xznhx43d6xdwudiuv7xcutpx6ympzybi/extras/demo_suite/bandwidthTest --mode=quick --memory=pinned --htod
[CUDA Bandwidth Test] - Starting...
Running on...

 Device 0: NVIDIA A100-PCIE-40GB
 Quick Mode

 Host to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     26127.6
```

PINNED is preferable to PAGEABLE

→ https://github.com/nvidia/cuda-samples

1_Utilities cont.

CUDA SDK cont.

```
cuda-zen sh@n3073-004: 1_Utilities$ /gpfs/opt/sw/cuda-zen/spack-0.19.0/opt/spack/linux-almalinux8-zen/gcc-12.2.0/cuda-12.9.0-xznhx43d6xdwudiuv7xcutpx6ympzybi/extras/demo_suite/bandwidthTest --help
cuda-zen sh@n3073-004: 1_Utilities$ /gpfs/opt/sw/cuda-zen/spack-0.19.0/opt/spack/linux-almalinux8-zen/gcc-12.2.0/cuda-12.9.0-xznhx43d6xdwudiuv7xcutpx6ympzybi/extras/demo_suite/bandwidthTest --mode=quick --memory=pageable --htod
[CUDA Bandwidth Test] - Starting...
Running on...

 Device 0: NVIDIA A100-PCIE-40GB
 Quick Mode

 Host to Device Bandwidth, 1 Device(s)
 PAGEABLE Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     16050.9

cuda-zen sh@n3073-004: 1_Utilities$ /gpfs/opt/sw/cuda-zen/spack-0.19.0/opt/spack/linux-almalinux8-zen/gcc-12.2.0/cuda-12.9.0-xznhx43d6xdwudiuv7xcutpx6ympzybi/extras/demo_suite/bandwidthTest --mode=quick --memory=pinned --htod
[CUDA Bandwidth Test] - Starting...
Running on...

 Device 0: NVIDIA A100-PCIE-40GB
 Quick Mode

 Host to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(MB/s)
   33554432                     26127.6
```

Why so far off the promised 1555 GB/s ???

PINNED is preferable to PAGEABLE

→ https://github.com/nvidia/cuda-samples

bandwidthTest.cu (essentials)

```
float
testDeviceToHostTransfer(unsigned int memSize, memoryMode memMode, bool wc)
{
    StopWatchInterface *timer = NULL;
    float elapsedTimeInMs = 0.0f;
    float bandwidthInGBs = 0.0f;
    unsigned char *h_idata = NULL;
    unsigned char *h_odata = NULL;
    cudaEvent_t start, stop;

    sdkCreateTimer(&timer);
    checkCudaErrors(cudaEventCreate(&start));
    checkCudaErrors(cudaEventCreate(&stop));

    //allocate host memory
    if (PINNED == memMode)
    {
        //pinned memory mode - use special function to get OS-pinned memory
        checkCudaErrors(cudaHostAlloc((void **)&h_idata, memSize, (wc) ? cudaHostAllocWriteCombined : 0));
        checkCudaErrors(cudaHostAlloc((void **)&h_odata, memSize, (wc) ? cudaHostAllocWriteCombined : 0));
    }
    else {
        //pageable memory mode - use malloc
    }
```

→ https://github.com/NVIDIA/cuda-samples/archive/refs/tags/v12.8.zip

bandwidthTest.cu (essentials)

```
float
testDeviceToHostTransfer(unsigned int memSize, memoryMode memMode, bool wc)
{
    StopWatchInterface *timer = NULL;
    float elapsedTimeInMs = 0.0f;
    float bandwidthInGBs = 0.0f;
    unsigned char *h_idata = NULL;
    unsigned char *h_odata = NULL;
    cudaEvent_t start, stop;

    sdkCreateTimer(&timer);
    checkCudaErrors(cudaEventCreate(&start));
    checkCudaErrors(cudaEventCreate(&stop));

    //allocate host memory
    if (PINNED == memMode)
    {
        //pinned memory mode - use special function to get OS-pinned memory
        checkCudaErrors(cudaHostAlloc((void **)&h_idata, memSize, (wc) ? cudaHostAllocWriteCombined : 0));
        checkCudaErrors(cudaHostAlloc((void **)&h_odata, memSize, (wc) ? cudaHostAllocWriteCombined : 0));
    }
    else {
        //pageable memory mode - use malloc
    }
```

32000000

→ https://github.com/NVIDIA/cuda-samples/archive/refs/tags/v12.8.zip

bandwidthTest.cu (essentials)

```
float
testDeviceToHostTransfer(unsigned int memSize, memoryMode memMode, bool wc)
{
    StopWatchInterface *timer = NULL;
    float elapsedTimeInMs = 0.0f;
    float bandwidthInGBs = 0.0f;
    unsigned char *h_idata = NULL;
    unsigned char *h_odata = NULL;
    cudaEvent_t start, stop;

    sdkCreateTimer(&timer);
    checkCudaErrors(cudaEventCreate(&start));
    checkCudaErrors(cudaEventCreate(&stop));

    //allocate host memory
    if (PINNED == memMode)
    {
        //pinned memory mode - use special function to get OS-pinned memory
        checkCudaErrors(cudaHostAlloc((void **)&h_idata, memSize, (wc) ? cudaHostAllocWriteCombined : 0));
        checkCudaErrors(cudaHostAlloc((void **)&h_odata, memSize, (wc) ? cudaHostAllocWriteCombined : 0));
    }
    else {
        //pageable memory mode - use malloc
    }
```

32000000

CUDA types

→ https://github.com/NVIDIA/cuda-samples/archive/refs/tags/v12.8.zip

bandwidthTest.cu (essentials)

```
float
testDeviceToHostTransfer(unsigned int memSize, memoryMode memMode, bool wc)
{
    StopWatchInterface *timer = NULL;
    float elapsedTimeInMs = 0.0f;
    float bandwidthInGBs = 0.0f;
    unsigned char *h_idata = NULL;
    unsigned char *h_odata = NULL;
    cudaEvent_t start, stop;

    sdkCreateTimer(&timer);
    checkCudaErrors(cudaEventCreate(&start));
    checkCudaErrors(cudaEventCreate(&stop));

    //allocate host memory
    if (PINNED == memMode)
    {
        //pinned memory mode - use special function to get OS-pinned memory
        checkCudaErrors(cudaHostAlloc((void **)&h_idata, memSize, (wc) ? cudaHostAllocWriteCombined : 0));
        checkCudaErrors(cudaHostAlloc((void **)&h_odata, memSize, (wc) ? cudaHostAllocWriteCombined : 0));
    }
    else {
        //pageable memory mode - use malloc
    }
```

32000000

CUDA types

CUDA time measurement

→ https://github.com/NVIDIA/cuda-samples/archive/refs/tags/v12.8.zip

bandwidthTest.cu (essentials)

```
float
testDeviceToHostTransfer(unsigned int memSize, memoryMode memMode, bool wc)
{
    StopWatchInterface *timer = NULL;
    float elapsedTimeInMs = 0.0f;
    float bandwidthInGBs = 0.0f;
    unsigned char *h_idata = NULL;
    unsigned char *h_odata = NULL;
    cudaEvent_t start, stop;

    sdkCreateTimer(&timer);
    checkCudaErrors(cudaEventCreate(&start));
    checkCudaErrors(cudaEventCreate(&stop));

    //allocate host memory
    if (PINNED == memMode)
    {
        //pinned memory mode - use special function to get OS-pinned memory
        checkCudaErrors(cudaHostAlloc((void **)&h_idata, memSize, (wc) ? cudaHostAllocWriteCombined : 0));
        checkCudaErrors(cudaHostAlloc((void **)&h_odata, memSize, (wc) ? cudaHostAllocWriteCombined : 0));
    }
    else {
        //pageable memory mode - use malloc
    }
```

32000000

CUDA types

CUDA time measurement

Error handling via encapsulation

→ https://github.com/NVIDIA/cuda-samples/archive/refs/tags/v12.8.zip

bandwidthTest.cu (essentials)

```
//initialize the memory
for (unsigned int i = 0; i < memSize/sizeof(unsigned char); i++)
{
    h_idata[i] = (unsigned char)(i & 0xff);
}
// allocate device memory
unsigned char *d_idata;
checkCudaErrors(cudaMalloc((void **) &d_idata, memSize));

//initialize the device memory
checkCudaErrors(cudaMemcpy(d_idata, h_idata, memSize,
                           cudaMemcpyHostToDevice));

//copy data from GPU to Host
sdkStartTimer(&timer);
checkCudaErrors(cudaEventRecord(start, 0));
if (PINNED == memMode)
{
    for (unsigned int i = 0; i < MEMCOPY_ITERATIONS; i++)
    {
        checkCudaErrors(cudaMemcpyAsync(h_odata, d_idata, memSize,
                                        cudaMemcpyDeviceToHost, 0));
    }
} else { ... }
```

→ https://github.com/NVIDIA/cuda-samples/archive/refs/tags/v12.8.zip

bandwidthTest.cu (essentials)

```
//initialize the memory
for (unsigned int i = 0; i < memSize/sizeof(unsigned char); i++)
{
    h_idata[i] = (unsigned char)(i & 0xff);
}
// allocate device memory
unsigned char *d_idata;
checkCudaErrors(cudaMalloc((void **) &d_idata, memSize));

//initialize the device memory
checkCudaErrors(cudaMemcpy(d_idata, h_idata, memSize,
                           cudaMemcpyHostToDevice));

//copy data from GPU to Host
sdkStartTimer(&timer);
checkCudaErrors(cudaEventRecord(start, 0));
if (PINNED == memMode)
{
    for (unsigned int i = 0; i < MEMCOPY_ITERATIONS; i++)
    {
        checkCudaErrors(cudaMemcpyAsync(h_odata, d_idata, memSize,
                                        cudaMemcpyDeviceToHost, 0));
    }
} else { ... }
```

Loop over 32M items

→ https://github.com/NVIDIA/cuda-samples/archive/refs/tags/v12.8.zip

bandwidthTest.cu (essentials)

```
//initialize the memory
for (unsigned int i = 0; i < memSize/sizeof(unsigned char); i++)
{
    h_idata[i] = (unsigned char)(i & 0xff);
}
// allocate device memory
unsigned char *d_idata;
checkCudaErrors(cudaMalloc((void **) &d_idata, memSize));

//initialize the device memory
checkCudaErrors(cudaMemcpy(d_idata, h_idata, memSize,
                           cudaMemcpyHostToDevice));

//copy data from GPU to Host
sdkStartTimer(&timer);
checkCudaErrors(cudaEventRecord(start, 0));
if (PINNED == memMode)
{
    for (unsigned int i = 0; i < MEMCOPY_ITERATIONS; i++)
    {
        checkCudaErrors(cudaMemcpyAsync(h_odata, d_idata, memSize,
                                        cudaMemcpyDeviceToHost, 0));
    }
} else { ... }
```

Loop over 32M items

bitwise add like (i % 255)

→ https://github.com/NVIDIA/cuda-samples/archive/refs/tags/v12.8.zip

bandwidthTest.cu (essentials)

```
//initialize the memory
for (unsigned int i = 0; i < memSize/sizeof(unsigned char); i++)
{
    h_idata[i] = (unsigned char)(i & 0xff);
}
// allocate device memory
unsigned char *d_idata;
checkCudaErrors(cudaMalloc((void **) &d_idata, memSize));

//initialize the device memory
checkCudaErrors(cudaMemcpy(d_idata, h_idata, memSize,
                           cudaMemcpyHostToDevice));

//copy data from GPU to Host
sdkStartTimer(&timer);
checkCudaErrors(cudaEventRecord(start, 0));
if (PINNED == memMode)
{
    for (unsigned int i = 0; i < MEMCOPY_ITERATIONS; i++)
    {
        checkCudaErrors(cudaMemcpyAsync(h_odata, d_idata, memSize,
                                        cudaMemcpyDeviceToHost, 0));
    }
} else { ... }
```

Loop over 32M items

bitwise add like (i % 255)

Memory set up on GPU

→ https://github.com/NVIDIA/cuda-samples/archive/refs/tags/v12.8.zip

bandwidthTest.cu (essentials)

```
//initialize the memory
for (unsigned int i = 0; i < memSize/sizeof(unsigned char); i++)
{
    h_idata[i] = (unsigned char)(i & 0xff);
}
// allocate device memory
unsigned char *d_idata;
checkCudaErrors(cudaMalloc((void **) &d_idata, memSize));

//initialize the device memory
checkCudaErrors(cudaMemcpy(d_idata, h_idata, memSize,
                           cudaMemcpyHostToDevice));

//copy data from GPU to Host
sdkStartTimer(&timer);
checkCudaErrors(cudaEventRecord(start, 0));
if (PINNED == memMode)
{
    for (unsigned int i = 0; i < MEMCOPY_ITERATIONS; i++)
    {
        checkCudaErrors(cudaMemcpyAsync(h_odata, d_idata, memSize,
                                        cudaMemcpyDeviceToHost, 0));
    }
} else { ... }
```

Loop over 32M items

bitwise add like (i % 255)

Memory set up on GPU

Timing begin

→ https://github.com/NVIDIA/cuda-samples/archive/refs/tags/v12.8.zip

bandwidthTest.cu (essentials)

```
//initialize the memory
for (unsigned int i = 0; i < memSize/sizeof(unsigned char); i++)
{
    h_idata[i] = (unsigned char)(i & 0xff);
}
// allocate device memory
unsigned char *d_idata;
checkCudaErrors(cudaMalloc((void **) &d_idata, memSize));

//initialize the device memory
checkCudaErrors(cudaMemcpy(d_idata, h_idata, memSize,
                           cudaMemcpyHostToDevice));

//copy data from GPU to Host
sdkStartTimer(&timer);
checkCudaErrors(cudaEventRecord(start, 0));
if (PINNED == memMode)
{
    for (unsigned int i = 0; i < MEMCOPY_ITERATIONS; i++)
    {
        checkCudaErrors(cudaMemcpyAsync(h_odata, d_idata, memSize,
                                        cudaMemcpyDeviceToHost, 0));
    }
} else { ... }
```

Loop over 32M items

bitwise add like (i % 255)

Memory set up on GPU

Timing begin

100

→ https://github.com/NVIDIA/cuda-samples/archive/refs/tags/v12.8.zip

bandwidthTest.cu (essentials)

```
    checkCudaErrors(cudaEventRecord(stop, 0));

    // make sure GPU has finished copying
    checkCudaErrors(cudaDeviceSynchronize());
    //get the total elapsed time in ms
    sdkStopTimer(&timer);
    checkCudaErrors(cudaEventElapsedTime(&elapsedTimeInMs, start, stop));

    //calculate bandwidth in GB/s
    double time_s = elapsedTimeInMs / 1e3;
    bandwidthInGBs = (memSize * (float)MEMCOPY_ITERATIONS) / (double)1e9;
    bandwidthInGBs = bandwidthInGBs / time_s;
    //clean up memory
    checkCudaErrors(cudaEventDestroy(stop));
    checkCudaErrors(cudaEventDestroy(start));
    sdkDeleteTimer(&timer);

    ... freeing allocated memory

    return bandwidthInGBs;
}
```

→ https://github.com/NVIDIA/cuda-samples/archive/refs/tags/v12.8.zip

bandwidthTest.cu (essentials)

```
    checkCudaErrors(cudaEventRecord(stop, 0));

    // make sure GPU has finished copying
    checkCudaErrors(cudaDeviceSynchronize());
    //get the total elapsed time in ms
    sdkStopTimer(&timer);
    checkCudaErrors(cudaEventElapsedTime(&elapsedTimeInMs, start, stop));

    //calculate bandwidth in GB/s
    double time_s = elapsedTimeInMs / 1e3;
    bandwidthInGBs = (memSize * (float)MEMCOPY_ITERATIONS) / (double)1e9;
    bandwidthInGBs = bandwidthInGBs / time_s;
    //clean up memory
    checkCudaErrors(cudaEventDestroy(stop));
    checkCudaErrors(cudaEventDestroy(start));
    sdkDeleteTimer(&timer);

    ... freeing allocated memory

    return bandwidthInGBs;
}
```

Timing end

→ https://github.com/NVIDIA/cuda-samples/archive/refs/tags/v12.8.zip

bandwidthTest.cu (essentials)

```
    checkCudaErrors(cudaEventRecord(stop, 0));

    // make sure GPU has finished copying
    checkCudaErrors(cudaDeviceSynchronize());
    //get the total elapsed time in ms
    sdkStopTimer(&timer);
    checkCudaErrors(cudaEventElapsedTime(&elapsedTimeInMs, start, stop));

    //calculate bandwidth in GB/s
    double time_s = elapsedTimeInMs / 1e3;
    bandwidthInGBs = (memSize * (float)MEMCOPY_ITERATIONS) / (double)1e9;
    bandwidthInGBs = bandwidthInGBs / time_s;
    //clean up memory
    checkCudaErrors(cudaEventDestroy(stop));
    checkCudaErrors(cudaEventDestroy(start));
    sdkDeleteTimer(&timer);

    ... freeing allocated memory

    return bandwidthInGBs;
}
```

Timing end

Exe time in ms

→ https://github.com/NVIDIA/cuda-samples/archive/refs/tags/v12.8.zip

bandwidthTest.cu (essentials)

```
    checkCudaErrors(cudaEventRecord(stop, 0));

    // make sure GPU has finished copying
    checkCudaErrors(cudaDeviceSynchronize());
    //get the total elapsed time in ms
    sdkStopTimer(&timer);
    checkCudaErrors(cudaEventElapsedTime(&elapsedTimeInMs, start, stop));

    //calculate bandwidth in GB/s
    double time_s = elapsedTimeInMs / 1e3;
    bandwidthInGBs = (memSize * (float)MEMCOPY_ITERATIONS) / (double)1e9;
    bandwidthInGBs = bandwidthInGBs / time_s;
    //clean up memory
    checkCudaErrors(cudaEventDestroy(stop));
    checkCudaErrors(cudaEventDestroy(start));
    sdkDeleteTimer(&timer);

    ... freeing allocated memory

    return bandwidthInGBs;
}
```

Timing end

Exe time in ms

Convert and compute bw

→ https://github.com/NVIDIA/cuda-samples/archive/refs/tags/v12.8.zip

- Data transfer between host and device is the slowest link involved in GPU computing
- Needs to be carefully designed/minimized case-by-case
- Peak bandwidth disparity between device memory $\leftrightarrow$ GPU cores (1555 GB/s on A100) and host memory $\leftrightarrow$ device memory (25 GB/s PCIe Gen4)
- GPU receives pinned memory only, which is a temporary translation of pageable host memory
- That's why the directly allocated memory in pinned form is transferred faster
- Another optimization strategy is to overlap memory transfer with computing

→ https://devblogs.nvidia.com/how-optimize-data-transfers-cuda-cc

- Efficiency comes with concurrently executing functions on all sorts of processing elements including CPU- and GPU-cores

→ https://devblogs.nvidia.com/gpu-pro-tip-cuda-7-streams-simplify-concurrency

0_Introduction/simpleStreams

CUDA SDK cont.

- Efficiency comes with concurrently executing functions on all sorts of processing elements including CPU- and GPU-cores
- CUDA applications manage concurrency with streams

→ https://devblogs.nvidia.com/gpu-pro-tip-cuda-7-streams-simplify-concurrency

0_Introduction/simpleStreams

CUDA SDK cont.

- Efficiency comes with concurrently executing functions on all sorts of processing elements including CPU- and GPU-cores
- CUDA applications manage concurrency with streams
- A stream is a sequence of commands executed in order

→ https://devblogs.nvidia.com/gpu-pro-tip-cuda-7-streams-simplify-concurrency

0_Introduction/simpleStreams

CUDA SDK cont.

- Efficiency comes with concurrently executing functions on all sorts of processing elements including CPU- and GPU-cores
- CUDA applications manage concurrency with streams
- A stream is a sequence of commands executed in order
- Several streams may execute their respective sequence of commands concurrently/asynchronously

→ https://devblogs.nvidia.com/gpu-pro-tip-cuda-7-streams-simplify-concurrency

- Efficiency comes with concurrently executing functions on all sorts of processing elements including CPU- and GPU-cores
- CUDA applications manage concurrency with streams
- A stream is a sequence of commands executed in order
- Several streams may execute their respective sequence of commands concurrently/asynchronously
- With CUDA 7 control over more than one (default stream) was introduced, so that multiple host threads can now have their own associated default stream for launching kernels

→ https://devblogs.nvidia.com/gpu-pro-tip-cuda-7-streams-simplify-concurrency

0_Introduction/simpleStreams

CUDA SDK cont.

- Efficiency comes with concurrently executing functions on all sorts of processing elements including CPU- and GPU-cores
- CUDA applications manage concurrency with streams
- A stream is a sequence of commands executed in order
- Several streams may execute their respective sequence of commands concurrently/asynchronously
- With CUDA 7 control over more than one (default stream) was introduced, so that multiple host threads can now have their own associated default stream for launching kernels
- Asynchronous commands in CUDA return control to the calling host thread before the device has finished the requested task (non-blocking), e.g. kernel launches, memory copies performed by functions with the Async suffix, etc.

→ https://devblogs.nvidia.com/gpu-pro-tip-cuda-7-streams-simplify-concurrency

CUDA streams

```
KrnlDmmy <<< numBlocks, threadsPerBlock, numBytes  >>> ();      // default stream
KrnlDmmy <<< numBlocks, threadsPerBlock, numBytes, 0  >>> ();   // stream 0
```

→ https://devblogs.nvidia.com/gpu-pro-tip-cuda-7-streams-simplify-concurrency
→ https://stackoverflow.com/questions/27162408/shared-memory-and-streams-when-launching-kernel

CUDA streams

Specifying a stream for a kernel launch (or memcopy) is optional

```
KrnlDmmy <<< numBlocks, threadsPerBlock, numBytes  >>> ();      // default stream
KrnlDmmy <<< numBlocks, threadsPerBlock, numBytes, 0  >>> ();   // stream 0
```

→ https://devblogs.nvidia.com/gpu-pro-tip-cuda-7-streams-simplify-concurrency
→ https://stackoverflow.com/questions/27162408/shared-memory-and-streams-when-launching-kernel

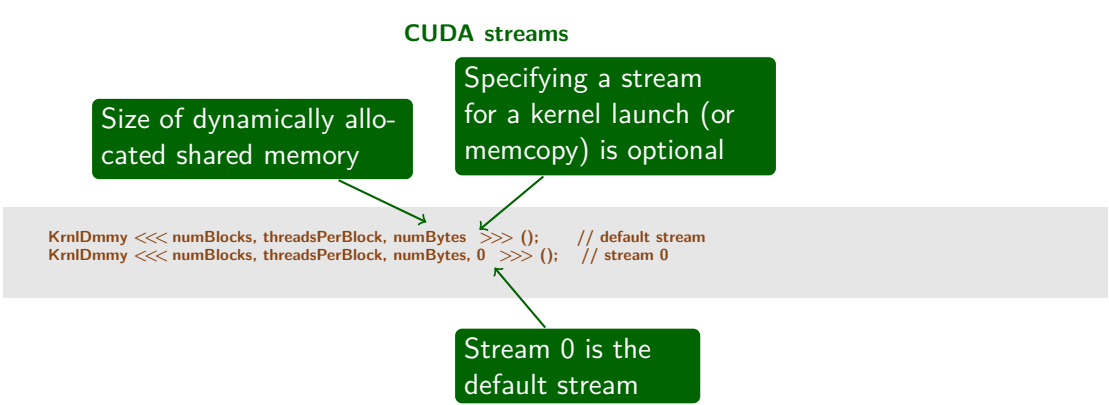
CUDA streams

Specifying a stream for a kernel launch (or memcopy) is optional

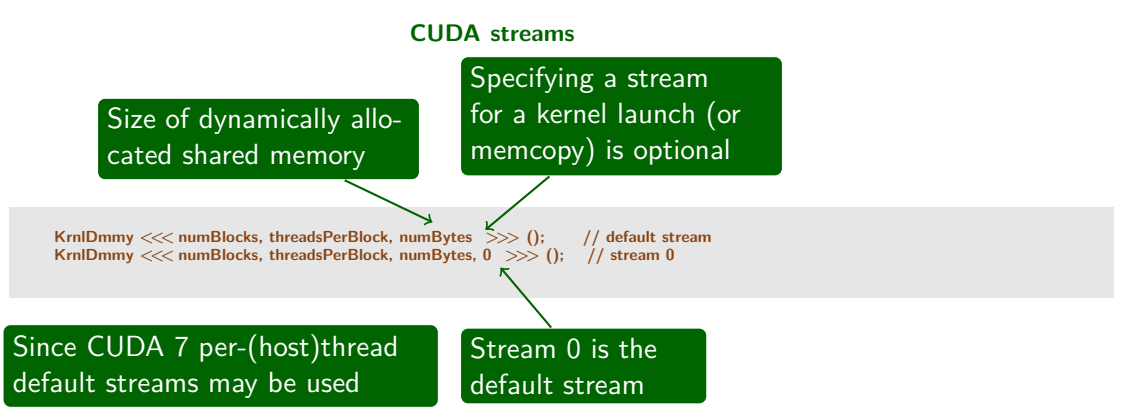
```
KrnlDmmy <<< numBlocks, threadsPerBlock, numBytes  >>> ();      // default stream
KrnlDmmy <<< numBlocks, threadsPerBlock, numBytes, 0  >>> ();   // stream 0
```

Stream 0 is the default stream

→ https://devblogs.nvidia.com/gpu-pro-tip-cuda-7-streams-simplify-concurrency
→ https://stackoverflow.com/questions/27162408/shared-memory-and-streams-when-launching-kernel



→ https://devblogs.nvidia.com/gpu-pro-tip-cuda-7-streams-simplify-concurrency
→ https://stackoverflow.com/questions/27162408/shared-memory-and-streams-when-launching-kernel



→ https://devblogs.nvidia.com/gpu-pro-tip-cuda-7-streams-simplify-concurrency
→ https://stackoverflow.com/questions/27162408/shared-memory-and-streams-when-launching-kernel

CUDA streams

Size of dynamically allocated shared memory

Specifying a stream for a kernel launch (or memcopy) is optional

```
KrnlDmmy <<< numBlocks, threadsPerBlock, numBytes >>> ();      // default stream
KrnlDmmy <<< numBlocks, threadsPerBlock, numBytes, 0 >>> ();   // stream 0
```

Since CUDA 7 per-(host)thread default streams may be used

Stream 0 is the default stream

Considered at compile time, nvcc --default-stream per-thread

→ https://devblogs.nvidia.com/gpu-pro-tip-cuda-7-streams-simplify-concurrency
→ https://stackoverflow.com/questions/27162408/shared-memory-and-streams-when-launching-kernel

0_Introduction/simpleStreams cont.

CUDA SDK cont.

```
const int N = 1048576;

__global__ void kernel(float *x, int n)
{
    int tid = blockIdx.x * blockDim.x + threadIdx.x;
    for (int i = tid; i <n; i += blockDim.x) {
        x[i] = sqrt(pow(3.14159,i));
    }
}
int main()
{
    const int num_streams = 8;
    cudaStream_t streams[num_streams];
    float *data[num_streams];
    for (int i = 0; i <num_streams; i++) {
        cudaStreamCreate(&streams[i]);
        cudaMalloc(&data[i], N * sizeof(float));
        // launch one worker kernel per stream
        kernel <<< 1, 64, 0, streams[i]  >>> (data[i], N);
        // launch a dummy kernel on the default stream
        kernel <<< 1, 1 >>> (0, 0);
    }
    cudaDeviceReset();
    return 0;
}
```

Thread-specific run through array x[] with stride blockDim.x

→ https://tinyurl.com/cudafordummies/ii/t/stream_test.cu

0_Introduction/simpleStreams cont.

CUDA SDK cont.

```
const int N = 1048576;

__global__ void kernel(float *x, int n)
{
    int tid = blockIdx.x * blockDim.x + threadIdx.x;
    for (int i = tid; i <n; i += blockDim.x) {
        x[i] = sqrt(pow(3.14159,i));
    }
}
int main()
{
    const int num_streams = 8;
    cudaStream_t streams[num_streams];
    float *data[num_streams];
    for (int i = 0; i <num_streams; i++) {
        cudaStreamCreate(&streams[i]);
        cudaMalloc(&data[i], N * sizeof(float));
        // launch one worker kernel per stream
        kernel <<< 1, 64, 0, streams[i]  >>> (data[i], N);
        // launch a dummy kernel on the default stream
        kernel <<< 1, 1 >>> (0, 0);
    }
    cudaDeviceReset();
    return 0;
}
```

Thread-specific run through array x[] with stride blockDim.x

Special type declaration

→ https://tinyurl.com/cudafordummies/ii/t/stream_test.cu

0_Introduction/simpleStreams cont.

CUDA SDK cont.

```
const int N = 1048576;

__global__ void kernel(float *x, int n)
{
    int tid = blockIdx.x * blockDim.x + threadIdx.x;
    for (int i = tid; i <n; i += blockDim.x) {
        x[i] = sqrt(pow(3.14159,i));
    }
}
int main()
{
    const int num_streams = 8;
    cudaStream_t streams[num_streams];
    float *data[num_streams];
    for (int i = 0; i <num_streams; i++) {
        cudaStreamCreate(&streams[i]);
        cudaMalloc(&data[i], N * sizeof(float));
        // launch one worker kernel per stream
        kernel <<< 1, 64, 0, streams[i]  >>> (data[i], N);
        // launch a dummy kernel on the default stream
        kernel <<< 1, 1 >>> (0, 0);
    }
    cudaDeviceReset();
    return 0;
}
```

Thread-specific run through array x[] with stride blockDim.x

Special type declaration

Stream creation and specific memory allocation

→ https://tinyurl.com/cudafordummies/ii/t/stream_test.cu

0_Introduction/simpleStreams cont.

CUDA SDK cont.

```
const int N = 1048576;

__global__ void kernel(float *x, int n)
{
    int tid = blockIdx.x * blockDim.x + threadIdx.x;
    for (int i = tid; i <n; i += blockDim.x) {
        x[i] = sqrt(pow(3.14159,i));
    }
}
int main()
{
    const int num_streams = 8;
    cudaStream_t streams[num_streams];
    float *data[num_streams];
    for (int i = 0; i <num_streams; i++) {
        cudaStreamCreate(&streams[i]);
        cudaMalloc(&data[i], N * sizeof(float));
        // launch one worker kernel per stream
        kernel <<< 1, 64, 0, streams[i]  >>> (data[i], N);
        // launch a dummy kernel on the default stream
        kernel <<< 1, 1 >>> (0, 0);
    }
    cudaDeviceReset();
    return 0;
}
```

Thread-specific run through array x[] with stride blockDim.x

Special type declaration

Stream creation and specific memory allocation

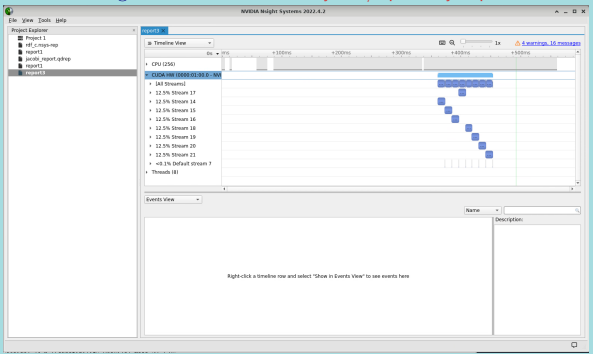
Kernel launch via streams

→ https://tinyurl.com/cudafordummies/ii/t/stream_test.cu

0_Introduction/simpleStreams cont.

CUDA SDK cont.

```
cuda–zen  sh@n3073–004: my_simpleStreams$   cp .../stream_test.cu ./simpleStreams.cu
cuda–zen  sh@n3073–004: my_simpleStreams$   mkdir build
cuda–zen  sh@n3073–004: my_simpleStreams$   cd build
cuda–zen  sh@n3073–004: build$   cmake ..
cuda–zen  sh@n3073–004: build$   make
cuda–zen  sh@n3073–004: build$   nsys nvprof ./simpleStreams
cuda–zen  sh@gui3068–009: build$   nsys-ui ./report1.nsys-rep
```

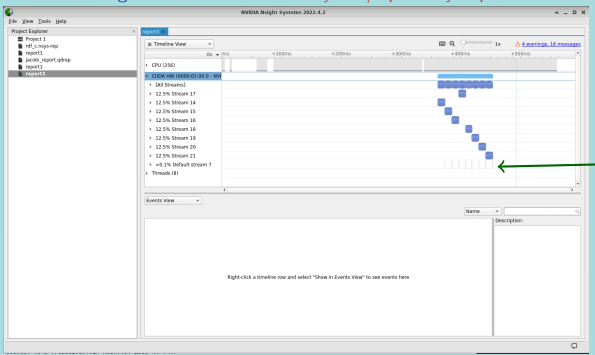


→ https://devblogs.nvidia.com/gpu-pro-tip-cuda-7-streams-simplify-concurrency

0_Introduction/simpleStreams cont.

CUDA SDK cont.

```
cuda–zen  sh@n3073–004: my_simpleStreams$   cp .../stream_test.cu ./simpleStreams.cu
cuda–zen  sh@n3073–004: my_simpleStreams$   mkdir build
cuda–zen  sh@n3073–004: my_simpleStreams$   cd build
cuda–zen  sh@n3073–004: build$   cmake ..
cuda–zen  sh@n3073–004: build$   make
cuda–zen  sh@n3073–004: build$   nsys nvprof ./simpleStreams
cuda–zen  sh@gui3068–009: build$   nsys-ui ./report1.nsys-rep
```



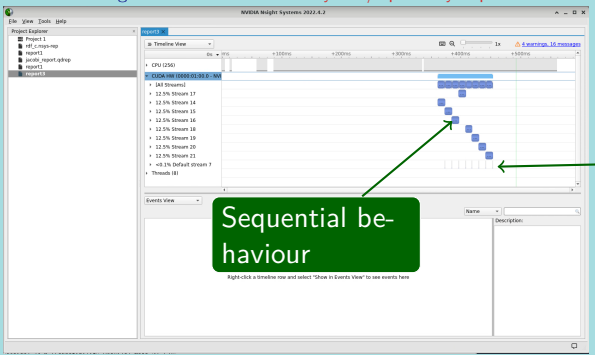
Interleaved dummy kernel sent to the default stream → no concurrency

→ https://devblogs.nvidia.com/gpu-pro-tip-cuda-7-streams-simplify-concurrency

0_Introduction/simpleStreams cont.

CUDA SDK cont.

```
cuda–zen  sh@n3073–004: my_simpleStreams$   cp .../stream_test.cu ./simpleStreams.cu
cuda–zen  sh@n3073–004: my_simpleStreams$   mkdir build
cuda–zen  sh@n3073–004: my_simpleStreams$   cd build
cuda–zen  sh@n3073–004: build$   cmake ..
cuda–zen  sh@n3073–004: build$   make
cuda–zen  sh@n3073–004: build$   nsys nvprof ./simpleStreams
cuda–zen  sh@gui3068–009: build$   nsys-ui ./report1.nsys-rep
```

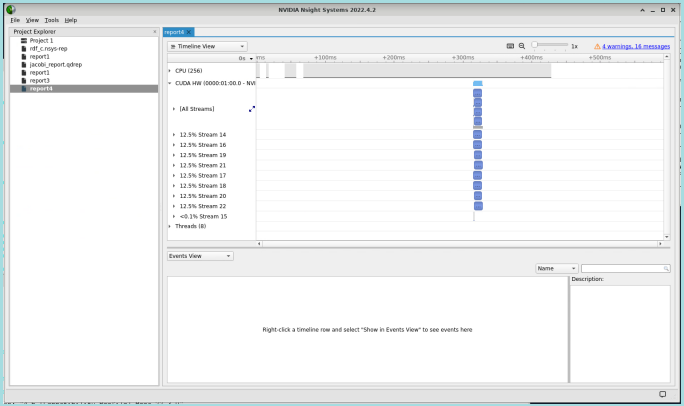


→ https://devblogs.nvidia.com/gpu-pro-tip-cuda-7-streams-simplify-concurrency

0_Introduction/simpleStreams cont.

CUDA SDK cont.

```
cuda–zen sh@n3073–004: build$   vi CMakeCache.txt # CMAKE_CUDA_FLAGS:STRING=--default-stream per-thread
cuda–zen sh@n3073–004: build$   make
cuda–zen sh@n3073–004: build$   nsys nvprof ./simpleStreams
cuda–zen sh@gui3068–009: build$   nsys-ui ./report2.nsys-rep
```

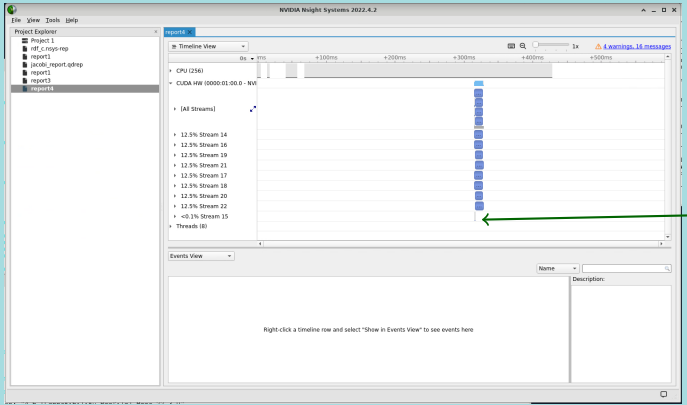


→ https://devblogs.nvidia.com/gpu-pro-tip-cuda-7-streams-simplify-concurrency

0_Introduction/simpleStreams cont.

CUDA SDK cont.

```
cuda–zen sh@n3073–004: build$   vi CMakeCache.txt # CMAKE_CUDA_FLAGS:STRING=--default-stream per-thread
cuda–zen sh@n3073–004: build$   make
cuda–zen sh@n3073–004: build$   nsys nvprof ./simpleStreams
cuda–zen sh@gui3068–009: build$   nsys-ui ./report2.nsys-rep
```



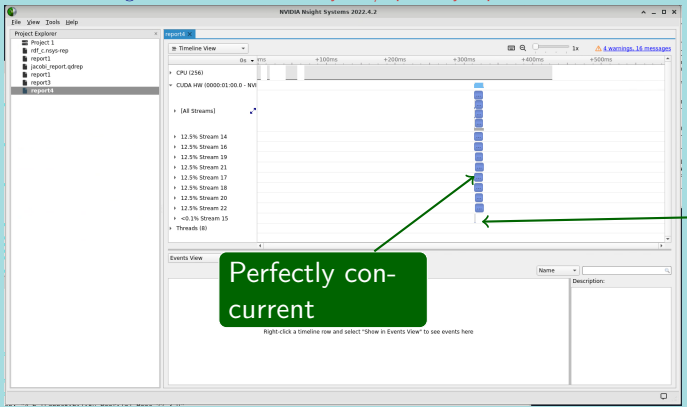
Dummy kernel (sent to the default stream) in parallel

→ https://devblogs.nvidia.com/gpu-pro-tip-cuda-7-streams-simplify-concurrency

0_Introduction/simpleStreams cont.

CUDA SDK cont.

```
cuda-zen sh@n3073-004: build$   vi CMakeCache.txt # CMAKE_CUDA_FLAGS:STRING=--default-stream per-thread
cuda-zen sh@n3073-004: build$   make
cuda-zen sh@n3073-004: build$   nsys nvprof ./simpleStreams
cuda-zen sh@gui3068-009: build$   nsys-ui ./report2.nsys-rep
```



Perfectly concurrent

Dummy kernel (sent to the default stream) in parallel

→ https://devblogs.nvidia.com/gpu-pro-tip-cuda-7-streams-simplify-concurrency

0_Introduction/simpleStreams cont.

CUDA SDK cont.

```
#include <omp.h>
const int N = 1048576;
__global__ void kernel(float *x, int n)
{
    int tid = blockIdx.x * blockDim.x + threadIdx.x;
    for (int i = tid; i <n; i += blockDim.x) {
        x[i] = sqrt(pow(3.14159,i));
    }
}
int main()
{
    const int num_streams = 8;
    cudaStream_t streams[num_streams];
    float *data[num_streams];
    omp_set_num_threads(num_streams);
    #pragma omp parallel for
    for (int i = 0; i <num_streams; i++) {
        cudaStreamCreate(&streams[i]);
        cudaMalloc(&data[i], N * sizeof(float));
        // launch one worker kernel per stream
        kernel <<< 1, 64, 0, streams[i]  >>> (data[i], N);
    }
    cudaDeviceReset();
    return 0;
}
```

Individual host-threads on separate CPU cores with associated stream

→ https://tinyurl.com/cudafordummies/ii/t/stream_test_v5.cu

0_Introduction/simpleStreams cont.

CUDA SDK cont.

```
#include <omp.h>
const int N = 1048576;
__global__ void kernel(float *x, int n)
{
    int tid = blockIdx.x * blockDim.x + threadIdx.x;
    for (int i = tid; i <n; i += blockDim.x) {
        x[i] = sqrt(pow(3.14159,i));
    }
}
int main()
{
    const int num_streams = 8;
    cudaStream_t streams[num_streams];
    float *data[num_streams];
    omp_set_num_threads(num_streams);
    #pragma omp parallel for
    for (int i = 0; i <num_streams; i++) {
        cudaStreamCreate(&streams[i]);
        cudaMalloc(&data[i], N * sizeof(float));
        // launch one worker kernel per stream
        kernel <<< 1, 64, 0, streams[i]  >>> (data[i], N);
    }
    cudaDeviceReset();
    return 0;
}
```

Individual host-threads on separate CPU cores with associated stream

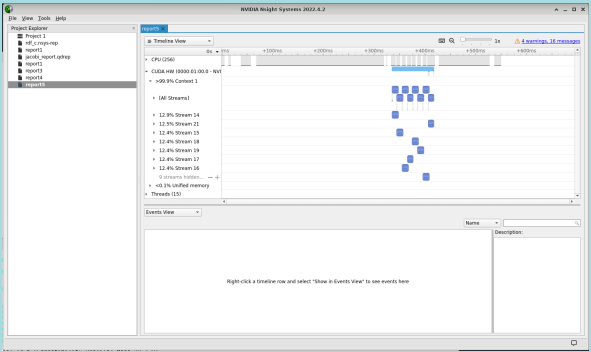
Simplest way of exploiting all available CPU/GPU resources

→ https://tinyurl.com/cudafordummies/ii/t/stream_test_v5.cu

0_Introduction/simpleStreams cont.

CUDA SDK cont.

```
cuda-zen sh@n3073-004: my_simpleStreams$   cp .../stream_test_v5.cu ./simpleStreams.cu
cuda-zen sh@n3073-004: my_simpleStreams$   cd build
cuda-zen sh@n3073-004: build$  vi CMakeCache.txt # CMAKE_CUDA_FLAGS:STRING=--default-stream per-thread -ccbin g++ -m64 -Xcompiler
-fopenmp
cuda-zen sh@n3073-004: build$   make
cuda-zen sh@n3073-004: build$   nsys nvprof ./simpleStreams
cuda-zen sh@gui3068-009: build$   nsys-ui ./report3.nsys-rep
```

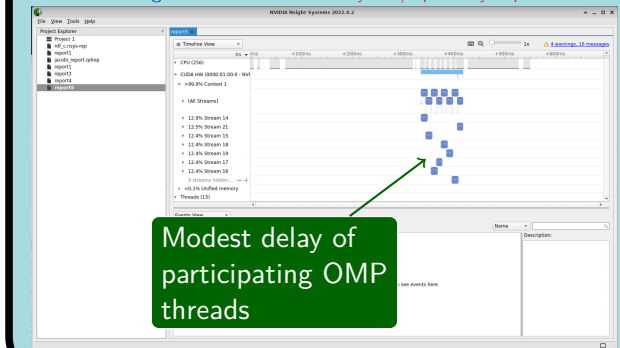


→ https://devblogs.nvidia.com/gpu-pro-tip-cuda-7-streams-simplify-concurrency

0_Introduction/simpleStreams cont.

CUDA SDK cont.

```
cuda-zen sh@n3073-004: my_simpleStreams$   cp .../stream_test_v5.cu ./simpleStreams.cu
cuda-zen sh@n3073-004: my_simpleStreams$   cd build
cuda-zen sh@n3073-004: build$  vi CMakeCache.txt # CMAKE_CUDA_FLAGS:STRING=--default-stream per-thread -ccbin g++ -m64 -Xcompiler
-fopenmp
cuda-zen sh@n3073-004: build$   make
cuda-zen sh@n3073-004: build$   nsys nvprof ./simpleStreams
cuda-zen sh@gui3068-009: build$   nsys-ui ./report3.nsys-rep
```

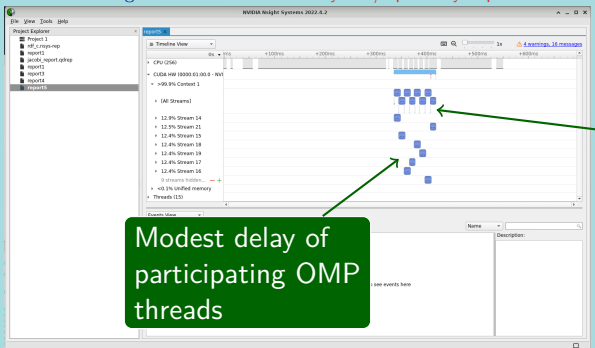


→ https://devblogs.nvidia.com/gpu-pro-tip-cuda-7-streams-simplify-concurrency

0_Introduction/simpleStreams cont.

CUDA SDK cont.

```
cuda–zen sh@n3073–004: my_simpleStreams$   cp .../stream_test_v5.cu ./simpleStreams.cu
cuda–zen sh@n3073–004: my_simpleStreams$   cd build
cuda–zen sh@n3073–004: build$  vi CMakeCache.txt # CMAKE_CUDA_FLAGS:STRING=--default-stream per-thread -ccbin g++ -m64 -Xcompiler
-fopenmp
cuda–zen sh@n3073–004: build$   make
cuda–zen sh@n3073–004: build$   nsys nvprof ./simpleStreams
cuda–zen sh@gui3068–009: build$   nsys-ui ./report3.nsys-rep
```



→ https://devblogs.nvidia.com/gpu-pro-tip-cuda-7-streams-simplify-concurrency

- Streams enable a lot of flexibilty in CUDA workloads
- Only the legacy default stream can pose problems
- Compiler flag --default-stream per-thread needs to be applied to all *.cu units involved
- cudaDeviceSynchronize() continues to synchronize everything on the device
- Individual streams may be synchronized via cudaStreamSynchronize()
- Ruling out interference by the default stream completely may be achieved with non-blocking streams, i.e. by passing the flag cudaStreamNonBlocking to cudaStreamCreate()

→ https://devblogs.nvidia.com/gpu-pro-tip-cuda-7-streams-simplify-concurrency

4_CUDA_Libraries/randomFog

CUDA SDK cont.

- CUDA is for graphics cards, so there are a lot of graphics examples too
- Example 4_CUDA_Libraries/randomFog
- Random number generation (200k) with CURAND
- Spherical polar coordinates are used (radius, rho, theta) normalized and presented as uniform distribution on the sphere
- Several options to display the data set

→ ~/cuda-samples/Samples/4_CUDA_Libraries/my_randomFog/build/randomFog

5_Domain_Specific/nbody

CUDA SDK cont.

- nbody is a CUDA demo of a gravitational n-body simulation
- Rather efficient scaling (strong) with multiple GPUs
- OpenGL rendering
- Command line args like -numbodies=10000 or -fp64 or -fullscreen or -cpu

→ ~/cuda-samples/Samples/5_Domain_Specific/my_nbody/build/nbody

4_CUDA_Libraries/oceanFFT

CUDA SDK cont.

- oceanFFT is a graphical demo of an ocean surface
- Height field is computed with the help of the CUFFT library (CUDA Fast Fourier Transform)
- OpenGL rendering
- 'w' — toggle wireframe

→ ~/cuda-samples/Samples/4_CUDA_Libraries/my_oceanFFT/build/oceanFFT

Take Home Messages

- CUDA SDK — a rich playground for beginners interested in learning the basics of GPU computing

Take Home Messages

- CUDA SDK — a rich playground for beginners interested in learning the basics of GPU computing
- Organized into different subject areas

Take Home Messages

- CUDA SDK — a rich playground for beginners interested in learning the basics of GPU computing
- Organized into different subject areas
- An exploratory space (memory management, error-handling...)

Take Home Messages

- CUDA SDK — a rich playground for beginners interested in learning the basics of GPU computing
- Organized into different subject areas
- An exploratory space (memory management, error-handling...)
- PCIe bandwidth remains a critical limitation in GPU computing

Take Home Messages

- CUDA SDK — a rich playground for beginners interested in learning the basics of GPU computing
- Organized into different subject areas
- An exploratory space (memory management, error-handling...)
- PCIe bandwidth remains a critical limitation in GPU computing
- CUDA streams bring in another level of flexibility, especially when run concurrently (perhaps from individual OpenMP threads on the host)

Take Home Messages

- CUDA SDK — a rich playground for beginners interested in learning the basics of GPU computing
- Organized into different subject areas
- An exploratory space (memory management, error-handling...)
- PCIe bandwidth remains a critical limitation in GPU computing
- CUDA streams bring in another level of flexibility, especially when run concurrently (perhaps from individual OpenMP threads on the host)
- Graphical demos — nice to have them too !