# CUDA SDK — Basic Concepts

Siegfried Höfinger

VSC Research Center, TU Wien

December 10, 2020

→ https://tinyurl.com/cuda4dummies/ii/l3/notes-l3.pdf

# Outline


Basic Conventions

0_Simple/simplePrintf

1_Utilities/bandwidthTest

0_Simple/simpleStreams

0_Simple/cudaOpenMP (variant)

7_CUDALibraries/randomFog

5_Simulations/nbody

5_Simulations/oceanFFT

Take Home Messages

# Basic Conventions
## CUDA SDK

- Automatically installed together with CUDA toolkit/driver
- 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 8 major subject areas
- Default location /usr/local/cuda/samples

→ https://docs.nvidia.com/cuda/cuda-samples/index.html#getting-started-with-cuda-samples

# Basic Conventions

## CUDA SDK cont.

```
[sh@n71-006]$ which nvcc
/opt/sw/x86_64/glibc-2.17/ivybridge-ep/cuda/9.1.85/bin/nvcc
[sh@n71-006]$ cd /opt/sw/x86_64/glibc-2.17/ivybridge-ep/cuda/9.1.85
[sh@n71-006]$ ls
bin      jre         nsightee_plugins         pkgconfig  tools
doc      lib64       NVIDIA_CUDA-9.1_Samples  samples    version.txt
extras   libnsight   nvml                     share
include  libnvvp     nvvm                     src
[sh@n71-006]$ cd samples
[sh@n71-006]$ ls
0_Simple     2_Graphics  4_Finance      6_Advanced       bin     EULA.txt
1_Utilities  3_Imaging   5_Simulations  7_CUDALibraries  common  Makefile
[sh@n71-006]$ cd 0_Simple
[sh@n71-006]$ ls
asyncAPI            matrixMul                  simpleLayeredTexture       simpleTemplates_nvrtc
cdpSimplePrint      matrixMulCUBLAS            simpleMPI                  simpleTexture
cdpSimpleQuicksort  matrixMulDrv               simpleMultiCopy            simpleTextureDrv
clock               matrixMul_nvrtc            simpleMultiGPU             simpleVoteIntrinsics
clock_nvrtc         simpleAssert               simpleOccupancy            simpleVoteIntrinsics_nvrtc
cppIntegration      simpleAssert_nvrtc         simpleP2P                  simpleZeroCopy
cppOverload         simpleAtomicIntrinsics     simplePitchLinearTexture   systemWideAtomics
cudaOpenMP          simpleAtomicIntrinsics_nvrtc  simplePrintf            template
cudaTensorCoreGemm  simpleCallback             simpleSeparateCompilation  UnifiedMemoryStreams
fp16ScalarProduct   simpleCooperativeGroups    simpleStreams              vectorAdd
inlinePTX           simpleCubemapTexture       simpleSurfaceWrite         vectorAddDrv
inlinePTX_nvrtc     simpleIPC                  simpleTemplates            vectorAdd_nvrtc
```

→ https://docs.nvidia.com/cuda/cuda-samples/index.html#getting-started-with-cuda-samples

# Basic Conventions

## CUDA SDK cont.

```
[sh@n71-006]$ which nvcc
/opt/sw/x86_64/glibc-2.17/ivybridge-ep/cuda/9.1.85/bin/nvcc
[sh@n71-006]$ cd /opt/sw/x86_64/glibc-2.17/ivybridge-ep/cuda/9.1.85
[sh@n71-006]$ ls
bin      jre         nsightee_plugins         pkgconfig  tools
doc      lib64       NVIDIA_CUDA-9.1_Samples  samples    version.txt
extras   libnsight   nvml                     share
include  libnvvp     nvvm                     src
[sh@n71-006]$ cd samples
[sh@n71-006]$ ls
0_Simple     2_Graphics  4_Finance      6_Advanced       bin     EULA.txt
1_Utilities  3_Imaging   5_Simulations  7_CUDALibraries  common  Makefile
[sh@n71-006]$ cd 0_Simple
[sh@n71-006]$ ls
asyncAPI            matrixMul                  simpleLayeredTexture       simpleTemplates_nvrtc
cdpSimplePrint      matrixMulCUBLAS            simpleMPI                  simpleTexture
cdpSimpleQuicksort  matrixMulDrv               simpleMultiCopy            simpleTextureDrv
clock               matrixMul_nvrtc            simpleMultiGPU             simpleVoteIntrinsics
clock_nvrtc         simpleAssert               simpleOccupancy            simpleVoteIntrinsics_nvrtc
cppIntegration      simpleAssert_nvrtc         simpleP2P                  simpleZeroCopy
cppOverload         simpleAtomicIntrinsics     simplePitchLinearTexture   systemWideAtomics
cudaOpenMP          simpleAtomicIntrinsics_nvrtc  simplePrintf            template
cudaTensorCoreGemm  simpleCallback             simpleSeparateCompilation  UnifiedMemoryStreams
fp16ScalarProduct   simpleCooperativeGroups    simpleStreams              vectorAdd
inlinePTX           simpleCubemapTexture       simpleSurfaceWrite         vectorAddDrv
inlinePTX_nvrtc     simpleIPC                  simpleTemplates            vectorAdd_nvrtc
```

→ https://docs.nvidia.com/cuda/cuda-samples/index.html#getting-started-with-cuda-samples

# Basic Conventions

## CUDA SDK cont.

| | |
|---|---|
| Simple | CUDA samples for beginners that illustrate key concepts with using CUDA and CUDA runtime APIs |
| Utilities | Utility samples that demonstrate how to query device capabilities and measure GPU/CPU bandwidth |
| Graphics | Graphical samples that demonstrate interoperability between CUDA and OpenGL or DirectX |
| Imaging | Samples that demonstrate image processing, compression, and data analysis |
| Finance | Samples that demonstrate parallel algorithms for financial computing |
| Simulations | Samples that illustrate a number of simulation algorithms implemented with CUDA |
| Advanced | Samples that illustrate advanced algorithms implemented with CUDA |
| CUDALibraries | Samples that illustrate how to use CUDA platform libraries, NPP, NVJPEG, NVGRAPH, cuBLAS, cuFFT, cuSPARSE, cuSOLVER and cuRAND |

→ https://docs.nvidia.com/cuda/cuda-samples/index.html#getting-started-with-cuda-samples

# Basic Conventions

## 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/cuda4dummies/ii/l3/smpl_assert.c

# Basic Conventions

## 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);
}
```

```
[sh@n71-006]$   gcc ./smpl_assert.c
[sh@n71-006]$   ./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/cuda4dummies/ii/l3/smpl_assert.c

# Basic Conventions

## 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.

```
[sh@n71-006]$   gcc ./smpl_assert.c
[sh@n71-006]$   ./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/cuda4dummies/ii/l3/smpl_assert.c

# Basic Conventions

## CUDA SDK cont.

There is also a CUDA sample in 0_Simple/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.

```
[sh@n71-006]$   gcc ./smpl_assert.c
[sh@n71-006]$   ./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/cuda4dummies/ii/l3/smpl_assert.c

# Basic Conventions

## CUDA SDK cont. — 0_Simple/simpleAssert

```
[sh@n71-006]$  cd /opt/sw/x86_64/glibc-2.17/ivybridge-ep/cuda/9.1.85/samples/0_Simple/simpleAssert
[sh@n71-006]$  ls
Makefile  NsightEclipse.xml  readme.txt  simpleAssert.cu
[sh@n71-006]$  mkdir -p $HOME/myplayground
[sh@n71-006]$  cp ./* $HOME/myplayground/
[sh@n71-006]$  cd $HOME/myplayground/
[sh@n71-006]$  make
/opt/src/USIF/cuda/9.1.85/install/bin/nvcc -ccbin g++ -I../../common/inc  -m64    -gencode
arch=compute_30,code=sm_30 -gencode arch=compute_35,code=sm_35 -gencode arch=compute_37,code=sm_37
-gencode arch=compute_50,code=sm_50 -gencode arch=compute_52,code=sm_52 -gencode
arch=compute_60,code=sm_60 -gencode arch=compute_61,code=sm_61 -gencode arch=compute_70,code=sm_70
-gencode arch=compute_70,code=compute_70 -o simpleAssert.o -c simpleAssert.cu

simpleAssert.cu:27:75: fatal error: helper_functions.h: No such file or directory
 #include <helper_functions.h>    // includes cuda.h and cuda_runtime_api.h
                                                                           ^

compilation terminated.
make: *** [simpleAssert.o] Error 1
```

→ https://docs.nvidia.com/cuda/cuda-samples/index.html#getting-started-with-cuda-samples

# Basic Conventions

## CUDA SDK cont. — 0_Simple/simpleAssert

```
[sh@n71-006]$  cd /opt/sw/x86_64/glibc-2.17/ivybridge-ep/cuda/9.1.85/samples/0_Simple/simpleAssert
[sh@n71-006]$  ls
Makefile  NsightEclipse.xml  readme.txt  simpleAssert.cu
[sh@n71-006]$  mkdir -p $HOME/myplayground
[sh@n71-006]$  cp ./* $HOME/myplayground/
[sh@n71-006]$  cd $HOME/myplayground/
[sh@n71-006]$  make
/opt/src/USIF/cuda/9.1.85/install/bin/nvcc -ccbin g++ -I../../common/inc  -m64    -gencode
arch=compute_30,code=sm_30 -gencode arch=compute_35,code=sm_35 -gencode arch=compute_37,code=sm_37
-gencode arch=compute_50,code=sm_50 -gencode arch=compute_52,code=sm_52 -gencode
arch=compute_60,code=sm_60 -gencode arch=compute_61,code=sm_61 -gencode arch=compute_70,code=sm_70
-gencode arch=compute_70,code=compute_70 -o simpleAssert.o -c simpleAssert.cu

simpleAssert.cu:27:75: fatal error: helper_functions.h: No such file or directory
 #include <helper_functions.h>    // includes cuda.h and cuda_runtime_api.h
                                                                           ^

compilation terminated.
make: *** [simpleAssert.o] Error 1
```

What's wrong here ? Can't even the simplest things work as expected ?

→ https://docs.nvidia.com/cuda/cuda-samples/index.html#getting-started-with-cuda-samples

# Basic Conventions

## CUDA SDK cont. — 0_Simple/simpleAssert

```
[sh@n71-006]$  cd /opt/sw/x86_64/glibc-2.17/ivybridge-ep/cuda/9.1.85/samples/0_Simple/simpleAssert
[sh@n71-006]$  ls
Makefile  NsightEclipse.xml  readme.txt  simpleAssert.cu
[sh@n71-006]$  mkdir -p $HOME/myplayground
[sh@n71-006]$  cp ./* $HOME/myplayground/
[sh@n71-006]$  cd $HOME/myplayground/
[sh@n71-006]$  make
/opt/src/USIF/cuda/9.1.85/install/bin/nvcc -ccbin g++ -I../../common/inc  -m64    -gencode
arch=compute_30,code=sm_30 -gencode arch=compute_35,code=sm_35 -gencode arch=compute_37,code=sm_37
-gencode arch=compute_50,code=sm_50 -gencode arch=compute_52,code=sm_52 -gencode
arch=compute_60,code=sm_60 -gencode arch=compute_61,code=sm_61 -gencode arch=compute_70,code=sm_70
-gencode arch=compute_70,code=compute_70 -o simpleAssert.o -c simpleAssert.cu

simpleAssert.cu:27:75: fatal error: helper_functions.h: No such file or directory
 #include <helper_functions.h>    // includes cuda.h and cuda_runtime_api.h
                                                                           ^

compilation terminated.
make: *** [simpleAssert.o] Error 1
```

What's wrong here ? Can't even the simplest things work as expected ?

Just a header file is missing

→ https://docs.nvidia.com/cuda/cuda-samples/index.html#getting-started-with-cuda-samples

# Basic Conventions

## CUDA SDK cont. — 0_Simple/simpleAssert

```
[sh@n71-006]$ ls /opt/sw/x86_64/glibc-2.17/ivybridge-ep/cuda/9.1.85/samples/common/inc
drvapi_error_string.h  helper_image.h         nvShaderUtils.h
dynlink_d3d10.h        helper_math.h          nvVector.h
dynlink_d3d11.h        helper_multiprocess.h  paramgl.h
exception.h            helper_string.h        param.h
GL                     helper_timer.h         rendercheck_d3d10.h
helper_cuda_drvapi.h   multithreading.h       rendercheck_d3d11.h
helper_cuda.h          nvMath.h               rendercheck_d3d9.h
helper_cusolver.h      nvMatrix.h             rendercheck_gles.h
helper_functions.h     nvQuaternion.h         rendercheck_gl.h
helper_gl.h            nvrtc_helper.h         timer.h

[sh@n71-006]$ vi Makefile

[sh@n71-006]$ make
/opt/src/USIF/cuda/9.1.85/install/bin/nvcc -ccbin g++ -I/opt/src/USIF/cuda/9.1.85/install/samples/common/inc
 -m64    -gencode arch=compute_30,code=sm_30 -gencode arch=compute_35,code=sm_35 -gencode arch=compute_37,code=
-gencode arch=compute_50,code=sm_50 -gencode arch=compute_52,code=sm_52 -gencode arch=compute_60,code=sm_60
-gencode arch=compute_61,code=sm_61 -gencode arch=compute_70,code=sm_70 -gencode arch=compute_70,code=compute_
-o simpleAssert.o -c simpleAssert.cu
/opt/src/USIF/cuda/9.1.85/install/bin/nvcc -ccbin g++ -I/opt/src/USIF/cuda/9.1.85/install/samples/common/inc
 -m64    -gencode arch=compute_30,code=sm_30 -gencode arch=compute_35,code=sm_35 -gencode arch=compute_37,code=
-gencode arch=compute_50,code=sm_50 -gencode arch=compute_52,code=sm_52 -gencode arch=compute_60,code=sm_60
-gencode arch=compute_61,code=sm_61 -gencode arch=compute_70,code=sm_70 -gencode arch=compute_70,code=compute_
-o simpleAssert simpleAssert.o
mkdir -p bin/x86_64/linux/release
cp simpleAssert bin/x86_64/linux/release
```

# Basic Conventions

## CUDA SDK cont. — 0_Simple/simpleAssert

```
[sh@n71-006]$ ls /opt/sw/x86_64/glibc-2.17/ivybridge-ep/cuda/9.1.85/samples/common/inc
drvapi_error_string.h  helper_image.h         nvShaderUtils.h
dynlink_d3d10.h        helper_math.h          nvVector.h
dynlink_d3d11.h        helper_multiprocess.h  paramgl.h
exception.h            helper_string.h        param.h
GL                     helper_timer.h         rendercheck_d3d10.h
helper_cuda_drvapi.h   multithreading.h       rendercheck_d3d11.h
helper_cuda.h          nvMath.h               rendercheck_d3d9.h
helper_cusolver.h      nvMatrix.h             rendercheck_gles.h
helper_functions.h     nvQuaternion.h         rendercheck_gl.h
helper_gl.h            nvrtc_helper.h         timer.h

[sh@n71-006]$ vi Makefile

[sh@n71-006]$ make
/opt/src/USIF/cuda/9.1.85/ins                                                    ommon/inc
 -m64    -gencode arch=compute_30,code=sm_30 -gencode arch=compute_35,code=sm_35 -gencode arch=compute_37,code=
-gencode arch=compute_50,code=sm_50 -gencode arch=compute_52,code=sm_52 -gencode arch=compute_60,code=sm_60
-gencode arch=compute_61,code=sm_61 -gencode arch=compute_70,code=sm_70 -gencode arch=compute_70,code=compute_
-o simpleAssert.o -c simpleAssert.cu
/opt/src/USIF/cuda/9.1.85/install/bin/nvcc -ccbin g++ -I/opt/src/USIF/cuda/9.1.85/install/samples/common/inc
 -m64    -gencode arch=compute_30,code=sm_30 -gencode arch=compute_35,code=sm_35 -gencode arch=compute_37,code=
-gencode arch=compute_50,code=sm_50 -gencode arch=compute_52,code=sm_52 -gencode arch=compute_60,code=sm_60
-gencode arch=compute_61,code=sm_61 -gencode arch=compute_70,code=sm_70 -gencode arch=compute_70,code=compute_
-o simpleAssert simpleAssert.o
mkdir -p bin/x86_64/linux/release
cp simpleAssert bin/x86_64/linux/release
```

Assign INCLUDES := the above directory

Drop ../../ in the final $(EXEC) lines

# Basic Conventions

## CUDA SDK cont. — 0_Simple/simpleAssert

```
[sh@n71-006]$ ./simpleAssert

simpleAssert starting...
OS_System_Type.release = 3.10.0-693.el7.x86_64
OS Info: <#1 SMP Tue Aug 22 21:09:27 UTC 2017>

GPU Device 0: "GeForce GTX 1080" with compute capability 6.1

Launch kernel to generate assertion failures

-- Begin assert output

simpleAssert.cu:47: void testKernel(int): block: [1,0,0], thread: [28,0,0] Assertion `gtid < N` failed.
simpleAssert.cu:47: void testKernel(int): block: [1,0,0], thread: [29,0,0] Assertion `gtid < N` failed.
simpleAssert.cu:47: void testKernel(int): block: [1,0,0], thread: [30,0,0] Assertion `gtid < N` failed.
simpleAssert.cu:47: 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://docs.nvidia.com/cuda/cuda-samples/index.html#getting-started-with-cuda-samples

# Basic Conventions

## CUDA SDK cont. — 0_Simple/simpleAssert

**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

## CUDA SDK cont. — 0_Simple/simpleAssert

**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 global threadID is $< 60$

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

# Basic Conventions

## CUDA SDK cont. — 0_Simple/simpleAssert

**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 global threadID is $< 60$

2 threadblocks of size 32

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

# Basic Conventions

## CUDA SDK cont. — 0_Simple/simpleAssert

**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 global threadID is $< 60$

2 threadblocks of size 32

C++ style of initializing variables dimGrid and dimBlock

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

# Basic Conventions

## CUDA SDK cont. — 0_Simple/simpleAssert

**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 global threadID is $< 60$

2 threadblocks of size 32

C++ style of initializing variables dimGrid and dimBlock

kernel launch

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

# Basic Conventions

## CUDA SDK cont. — 0_Simple/simpleAssert

Device behaviour identical to host; lots of useful information !

**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 global threadID is $< 60$

2 threadblocks of size 32

C++ style of initializing variables dimGrid and dimBlock

kernel launch

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

# Basic Conventions

## CUDA SDK cont. — 0_Simple/simpleAssert

**simpleAssert.cu**

Device behaviour identical to host; lots of useful information !

New type (enum) for error variable

```
__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 global threadID is $< 60$

2 threadblocks of size 32

C++ style of initializing variables dimGrid and dimBlock

kernel launch

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

# Basic Conventions

## CUDA SDK cont. — 0_Simple/simpleAssert

**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 global threadID is $< 60$

2 threadblocks of size 32

C++ style of initializing variables dimGrid and dimBlock

kernel launch

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

# Basic Conventions

## CUDA SDK cont. — 0_Simple/simpleAssert

### 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

# Basic Conventions

## CUDA SDK cont. — 0_Simple/simpleAssert

### 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

# Basic Conventions

## CUDA SDK cont. — 0_Simple/simpleAssert

### 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

# Basic Conventions

## CUDA SDK cont. — 0_Simple/simpleAssert

**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

# Basic Conventions

## CUDA SDK cont. — 0_Simple/simpleAssert

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

→ https://devblogs.nvidia.com/unified-memory-cuda-beginners

# Basic Conventions

## CUDA SDK cont. — 0_Simple/simpleAssert

- 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://devblogs.nvidia.com/unified-memory-cuda-beginners

# Basic Conventions

## CUDA SDK cont. — 0_Simple/simpleAssert

- 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://devblogs.nvidia.com/unified-memory-cuda-beginners

# Basic Conventions

## CUDA SDK cont. — 0_Simple/simpleAssert

- 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://devblogs.nvidia.com/unified-memory-cuda-beginners

# Basic Conventions

## CUDA SDK cont. — 0_Simple/simpleAssert

```
[sh@n71-006]$  ./simpleAssert

simpleAssert starting...
OS_System_Type.release = 3.10.0-693.el7.x86_64
OS Info: <#1 SMP Tue Aug 22 21:09:27 UTC 2017>

GPU Device 0: "GeForce GTX 1080" with compute capability 6.1

Launch kernel to generate assertion failures

-- Begin assert output

*** message from thread 0 ***
*** message from thread 1 ***
*** message from thread 2 ***
*** message from thread 3 ***
......
*** message from thread 29 ***
*** message from thread 30 ***
*** message from thread 31 ***
simpleAssert.cu:47: void testKernel(int): block: [1,0,0], thread: [28,0,0] Assertion `gtid < N` failed.
simpleAssert.cu:47: void testKernel(int): block: [1,0,0], thread: [29,0,0] Assertion `gtid < N` failed.
simpleAssert.cu:47: void testKernel(int): block: [1,0,0], thread: [30,0,0] Assertion `gtid < N` failed.
simpleAssert.cu:47: void testKernel(int): block: [1,0,0], thread: [31,0,0] Assertion `gtid < N` failed.

-- End assert output
```

→ https://docs.nvidia.com/cuda/cuda-samples/index.html#getting-started-with-cuda-samples

# 0_Simple/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://devblogs.nvidia.com/unified-memory-cuda-beginners

# 0_Simple/simplePrintf

## CUDA SDK cont.

```
[sh@n71-006]$  cp /opt/sw/x86_64/glibc-2.17/ivybridge-ep/cuda/9.1.85/samples/0_Simple/simplePrintf/* ./
[sh@n71-006]$  vi Makefile
[sh@n71-006]$  make
[sh@n71-006]$  ./simplePrintf
GPU Device 0: "GeForce GTX 1080" with compute capability 6.1

Device 0: "GeForce GTX 1080" with Compute 6.1 capability
printf() is called. Output:

[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
[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
```

→ https://docs.nvidia.com/cuda/cuda-samples/index.html#getting-started-with-cuda-samples

# 0_Simple/simplePrintf

## CUDA SDK cont.

```
[sh@n71-006]$  cp /opt/sw/x86_64/glibc-2.17/ivybridge-ep/cuda/9.1.85/samples/0_Simple/simplePrintf/* ./
[sh@n71-006]$  vi Makefile
[sh@n71-006]$  make
[sh@n71-006]$  ./simplePrintf
GPU Device 0: "GeForce GTX 1080" with compute capability 6.1

Device 0: "GeForce GTX 1080" with Compute 6.1 capability
printf() is called. Output:

[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
[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
```

set INCLUDES := and delete ../../ as before

→ https://docs.nvidia.com/cuda/cuda-samples/index.html#getting-started-with-cuda-samples

# 0_Simple/simplePrintf

## CUDA SDK cont.

```
[sh@n71-006]$  cp /opt/sw/x86_64/glibc-2.17/ivybridge-ep/cuda/9.1.85/samples/0_Simple/simplePrintf/* ./
[sh@n71-006]$  vi Makefile
[sh@n71-006]$  make
[sh@n71-006]$  ./simplePrintf
GPU Device 0: "GeForce GTX 1080" with compute capability 6.1

Device 0: "GeForce GTX 1080" with Compute 6.1 capability
printf() is called. Output:

[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
[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
```

set INCLUDES := and delete ../../ as before

4 x blocks with indices from 0-7

→ https://docs.nvidia.com/cuda/cuda-samples/index.html#getting-started-with-cuda-samples

# 0_Simple/simplePrintf

## CUDA SDK cont.

```
[sh@n71-006]$  cp /opt/sw/x86_64/glibc-2.17/ivybridge-ep/cuda/9.1.85/samples/0_Simple/simplePrintf/* ./
[sh@n71-006]$  vi Makefile
[sh@n71-006]$  make
[sh@n71-006]$  ./simplePrintf
GPU Device 0: "GeForce GTX 1080" with compute capability 6.1

Device 0: "GeForce GTX 1080" with Compute 6.1 capability
printf() is called. Output:

[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
[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
```

set INCLUDES := and delete ../../ as before

4 x blocks with indices from 0-7

stochastic order of blocks

→ https://docs.nvidia.com/cuda/cuda-samples/index.html#getting-started-with-cuda-samples

# 0_Simple/simplePrintf

## 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://docs.nvidia.com/cuda/cuda-samples/index.html#getting-started-with-cuda-samples

# 0_Simple/simplePrintf

## 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

→ https://docs.nvidia.com/cuda/cuda-samples/index.html#getting-started-with-cuda-samples

# 0_Simple/simplePrintf

## 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

→ https://docs.nvidia.com/cuda/cuda-samples/index.html#getting-started-with-cuda-samples

# 0_Simple/simplePrintf

## 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://docs.nvidia.com/cuda/cuda-samples/index.html#getting-started-with-cuda-samples

# 0_Simple/simplePrintf

## CUDA SDK cont.

- 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://docs.nvidia.com/cuda/cuda-samples/index.html#getting-started-with-cuda-samples

# 1_Utilities/bandwidthTest

## CUDA SDK cont.

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

→ https://docs.nvidia.com/cuda/cuda-samples/index.html#getting-started-with-cuda-samples

# 1_Utilities/bandwidthTest

## CUDA SDK cont.

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

→ https://docs.nvidia.com/cuda/cuda-samples/index.html#getting-started-with-cuda-samples

# 1_Utilities/bandwidthTest

## CUDA SDK cont.

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

→ https://docs.nvidia.com/cuda/cuda-samples/index.html#getting-started-with-cuda-samples

# 1_Utilities/bandwidthTest

## CUDA SDK cont.

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

→ https://docs.nvidia.com/cuda/cuda-samples/index.html#getting-started-with-cuda-samples

# 1_Utilities/bandwidthTest

## CUDA SDK cont.

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

→ https://docs.nvidia.com/cuda/cuda-samples/index.html#getting-started-with-cuda-samples

# 1_Utilities/bandwidthTest

## CUDA SDK cont.

```
[sh@n71-006]$  cp /opt/sw/x86_64/glibc-2.17/ivybridge-ep/cuda/9.1.85/samples/1_Utilities/bandwidthTest/* ./
[sh@n71-006]$  vi Makefile
[sh@n71-006]$  make
[sh@n71-006]$  ./bandwidthTest
[CUDA Bandwidth Test] - Starting...
Running on...

 Device 0: GeForce GTX 1080
 Quick Mode

 Host to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(GB/s)
   32000000                     13.1

 Device to Host Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(GB/s)
   32000000                     13.2

 Device to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(GB/s)
   32000000                     236.1

Result = PASS
```

→ https://docs.nvidia.com/cuda/cuda-samples/index.html#getting-started-with-cuda-samples

# 1_Utilities/bandwidthTest

## CUDA SDK cont.

```
[sh@n71-006]$  cp /opt/sw/x86_64/glibc-2.17/ivybridge-ep/cuda/9.1.85/samples/1_Utilities/bandwidthTest/* ./
[sh@n71-006]$  vi Makefile
[sh@n71-006]$  make
[sh@n71-006]$  ./bandwidthTest
[CUDA Bandwidth Test] - Starting...
Running on...

 Device 0: GeForce GTX 1080
 Quick Mode

 Host to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(GB/s)
   32000000                     13.1

 Device to Host Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(GB/s)
   32000000                     13.2

 Device to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(GB/s)
   32000000                     236.1

Result = PASS
```

INCLUDES := as previously and ../../ to be dropped again

→ https://docs.nvidia.com/cuda/cuda-samples/index.html#getting-started-with-cuda-samples

# 1_Utilities/bandwidthTest

## CUDA SDK cont.

```
[sh@n71-006]$  cp /opt/sw/x86_64/glibc-2.17/ivybridge-ep/cuda/9.1.85/samples/1_Utilities/bandwidthTest/* ./
[sh@n71-006]$  vi Makefile
[sh@n71-006]$  make
[sh@n71-006]$  ./bandwidthTest
[CUDA Bandwidth Test] - Starting...
Running on...

 Device 0: GeForce GTX 1080
 Quick Mode

 Host to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(GB/s)
   32000000                     13.1

 Device to Host Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(GB/s)
   32000000                     13.2

 Device to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(GB/s)
   32000000                     236.1

Result = PASS
```

INCLUDES := as previously and ../../ to be dropped again

32MB (pinned) with ≈ 13 GB/s

→ https://docs.nvidia.com/cuda/cuda-samples/index.html#getting-started-with-cuda-samples

# 1_Utilities/bandwidthTest

## CUDA SDK cont.

```
[sh@n71-006]$   ./bandwidthTest --help
[sh@n71-006]$   ./bandwidthTest --mode=shmoo
[CUDA Bandwidth Test] - Starting...
Running on...

 Device 0: GeForce GTX 1080
 Quick Mode

 Host to Device Bandwidth, 1 Device(s)                  Device to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers                                PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(GB/s)           Transfer Size (Bytes)        Bandwidth(GB/s)
   1000                         0.3                       1000                         0.4
   2000                         0.6                       2000                         0.8
   ......                                                 ......
   68000000                     13.1                      68000000                     238.9

.............................................       Result = PASS
 Device to Host Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(GB/s)
   1000                         0.3
   2000                         0.6
   ......
   68000000                     13.2

.............................................
```

→ https://docs.nvidia.com/cuda/cuda-samples/index.html#getting-started-with-cuda-samples

# 1_Utilities/bandwidthTest

## CUDA SDK cont.

```
[sh@n71-006]$   ./bandwidthTest --help
[sh@n71-006]$   ./bandwidthTest --mode=shmoo
[CUDA Bandwidth Test] - Starting...
Running on...

 Device 0: GeForce GTX 1080
 Quick Mode

 Host to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(GB/s)
   1000                         0.3
   2000                         0.6
   ......
   68000000                     13.1

 Device to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(GB/s)
   1000                         0.4
   2000                         0.8
   ......
   68000000                     238.9

.....................................      Result = PASS
 Device to Host Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(GB/s)
   1000                         0.3
   2000                         0.6
   ......
   68000000                     13.2

.....................................
```

Pretty much the values seen before

→ https://docs.nvidia.com/cuda/cuda-samples/index.html#getting-started-with-cuda-samples

# 1_Utilities/bandwidthTest

## CUDA SDK cont.

```
[sh@n71-006]$   ./bandwidthTest --help
[sh@n71-006]$   ./bandwidthTest --mode=quick --memory=pageable --htod

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

 Device 0: GeForce GTX 1080
 Quick Mode

 Host to Device Bandwidth, 1 Device(s)
 PAGEABLE Memory Transfers
   Transfer Size (Bytes)        Bandwidth(GB/s)
   32000000                     10.0

[sh@n71-006]$   ./bandwidthTest --mode=quick --memory=pinned --htod

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

 Device 0: GeForce GTX 1080
 Quick Mode

 Host to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(GB/s)
   32000000                     13.1
```

→ https://docs.nvidia.com/cuda/cuda-samples/index.html#getting-started-with-cuda-samples

# 1_Utilities/bandwidthTest

## CUDA SDK cont.

```
[sh@n71-006]$   ./bandwidthTest --help
[sh@n71-006]$   ./bandwidthTest --mode=quick --memory=pageable --htod

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

 Device 0: GeForce GTX 1080
 Quick Mode

 Host to Device Bandwidth, 1 Device(s)
 PAGEABLE Memory Transfers
   Transfer Size (Bytes)        Bandwidth(GB/s)
   32000000                     10.0

[sh@n71-006]$   ./bandwidthTest --mode=quick --memory=pinned --htod

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

 Device 0: GeForce GTX 1080
 Quick Mode

 Host to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(GB/s)
   32000000                     13.1
```

PINNED is slightly better for both htod and dtoh

→ https://docs.nvidia.com/cuda/cuda-samples/index.html#getting-started-with-cuda-samples

# 1_Utilities/bandwidthTest

## CUDA SDK cont.

```
[sh@n71-006]$   ./bandwidthTest --help
[sh@n71-006]$   ./bandwidthTest --mode=quick --memory=pageable --htod

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

 Device 0: GeForce GTX 1080
 Quick Mode

 Host to Device Bandwidth, 1 Device(s)
 PAGEABLE Memory Transfers
   Transfer Size (Bytes)        Bandwidth(GB/s)
   32000000                     10.0

[sh@n71-006]$   ./bandwidthTest --mode=quick --memory=pinned --htod

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

 Device 0: GeForce GTX 1080
 Quick Mode

 Host to Device Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(GB/s)
   32000000                     13.1
```

Why so far off the promised 320 GB/s ???

PINNED is slightly better for both htod and dtoh

→ https://docs.nvidia.com/cuda/cuda-samples/index.html#getting-started-with-cuda-samples

# 1_Utilities/bandwidthTest

## CUDA SDK cont.

```
[sh@n71-006]$  nvprof ./bandwidthTest --mode=quick --memory=pinned --dtoh

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

==24545== NVPROF is profiling process 24545, command: ./bandwidthTest --mode=quick --memory=pinned --dtoh
 Device 0: GeForce GTX 1080
 Quick Mode

 Device to Host Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(GB/s)
   32000000                     13.2

Result = PASS

NOTE: The CUDA Samples are not meant for performance measurements. Results may vary when GPU Boost is enab
==24545== Profiling application: ./bandwidthTest --mode=quick --memory=pinned --dtoh
==24545== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   99.01%  242.49ms       100  2.4249ms  2.4246ms  2.4305ms  [CUDA memcpy DtoH]
                    0.99%  2.4356ms         1  2.4356ms  2.4356ms  2.4356ms  [CUDA memcpy HtoD]
      API calls:   53.22%  242.40ms         1  242.40ms  242.40ms  242.40ms  cudaDeviceSynchronize
                   40.73%  185.50ms         2  92.751ms  1.9300us  185.50ms  cudaEventCreate
                    3.40%  15.475ms         2  7.7374ms  7.6508ms  7.8240ms  cudaHostAlloc
                    1.55%  7.0521ms         2  3.5261ms  3.4709ms  3.5812ms  cudaFreeHost
```

→ https://docs.nvidia.com/cuda/cuda-samples/index.html#getting-started-with-cuda-samples

# 1_Utilities/bandwidthTest

## CUDA SDK cont.

```
[sh@n71-006]$  nvprof ./bandwidthTest --mode=quick --memory=pinned --dtoh

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

==24545== NVPROF is profiling process 24545, command: ./bandwidthTest --mode=quick --memory=pinned --dtoh
 Device 0: GeForce GTX 1080
 Quick Mode

 Device to Host Bandwidth, 1 Device(s)
 PINNED Memory Transfers
   Transfer Size (Bytes)        Bandwidth(GB/s)
   32000000                     13.2

Result = PASS

NOTE: The CUDA Samples are not meant for performance measurements. Results may vary when GPU Boost is enab
==24545== Profiling application: ./bandwidthTest --mode=quick --memory=pinned --dtoh
==24545== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   99.01%  242.49ms       100  2.4249ms  2.4246ms  2.4305ms  [CUDA memcpy DtoH]
                    0.99%  2.4356ms         1  2.4356ms  2.4356ms  2.4356ms  [CUDA memcpy HtoD]
      API calls:   53.22%  242.40ms         1  242.40ms  242.40ms  242.40ms  cudaDeviceSynchronize
                   40.73%  185.50ms         2  92.751ms  1.9300us  185.50ms  cudaEventCreate
                    3.40%  15.475ms         2  7.7374ms  7.6508ms  7.8240ms  cudaHostAlloc
                    1.55%  7.0521ms         2  3.5261ms  3.4709ms  3.5812ms  cudaFreeHost
```

0.032 / 0.0024249 = 13.2 GB/s

→ https://docs.nvidia.com/cuda/cuda-samples/index.html#getting-started-with-cuda-samples

# 1_Utilities/bandwidthTest

## CUDA SDK cont.

**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_idata, memSize, (wc) ? cudaHostAllocWriteCombined : 0));
    }
    else
    {
        //pageable memory mode - use malloc
        ...
    }
```

# 1_Utilities/bandwidthTest

## CUDA SDK cont.

**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_idata, memSize, (wc) ? cudaHostAllocWriteCombined : 0));
    }
    else
    {
        //pageable memory mode - use malloc
        ...
    }
```

# 1_Utilities/bandwidthTest

## CUDA SDK cont.

**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_idata, memSize, (wc) ? cudaHostAllocWriteCombined : 0));
    }
    else
    {
        //pageable memory mode - use malloc
        ...
    }
```

32000000

0

0

CUDA types

# 1_Utilities/bandwidthTest

## CUDA SDK cont.

**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_idata, memSize, (wc) ? cudaHostAllocWriteCombined : 0));
    }
    else
    {
        //pageable memory mode - use malloc
        ...
    }
```

32000000

0

0

CUDA types

CUDA time measurement

# 1_Utilities/bandwidthTest

## CUDA SDK cont.

**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_idata, memSize, (wc) ? cudaHostAllocWriteCombined : 0));
    }
    else
    {
        //pageable memory mode - use malloc
        ...
    }
```

32000000

0

0

CUDA types

CUDA time measurement

Error handling via encapsulation

# 1_Utilities/bandwidthTest

## CUDA SDK cont.

**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 { ... }
```

# 1_Utilities/bandwidthTest

## CUDA SDK cont.

**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

# 1_Utilities/bandwidthTest

## CUDA SDK cont.

**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)

# 1_Utilities/bandwidthTest

## CUDA SDK cont.

**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

# 1_Utilities/bandwidthTest

## CUDA SDK cont.

**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

# 1_Utilities/bandwidthTest

## CUDA SDK cont.

**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

# 1_Utilities/bandwidthTest

## CUDA SDK cont.

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

# 1_Utilities/bandwidthTest

## CUDA SDK cont.

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

# 1_Utilities/bandwidthTest

## CUDA SDK cont.

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

# 1_Utilities/bandwidthTest

## CUDA SDK cont.

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

# 1_Utilities/bandwidthTest

## CUDA SDK cont.

- 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 (320 GB/s on Pascal) and host memory $\leftrightarrow$ device memory (16 GB/s PCIe 3.1)
- nvprof can be used here in a very straightforward way to get an idea of achievable transfer rates
- 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 slightly faster
- Another optimization strategy is to overlap memory transfers with computing

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

# 0_Simple/simpleStreams

## CUDA SDK cont.

- 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_Simple/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_Simple/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_Simple/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 sequences of commands concurrently/asynchronously

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

# 0_Simple/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 sequences 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_Simple/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 sequences 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

# 0_Simple/simpleStreams

## CUDA SDK cont.

### 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

# 0_Simple/simpleStreams

## CUDA SDK cont.

### 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

# 0_Simple/simpleStreams

## CUDA SDK cont.

### 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
```

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

# 0_Simple/simpleStreams

## CUDA SDK cont.

### 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

→ 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_Simple/simpleStreams

## CUDA SDK cont.

### 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_Simple/simpleStreams

## CUDA SDK cont.

```
const int N = 1048576;

__global__ void kernel(float *x, int n)
{
    int tid = threadIdx.x + blockIdx.x * blockDim.x;
    for (int i = tid; i <n; i += blockDim.x * gridDim.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/cuda4dummies/ii/t/stream_test.cu

# 0_Simple/simpleStreams

## CUDA SDK cont.

```
const int N = 1048576;

__global__ void kernel(float *x, int n)
{
    int tid = threadIdx.x + blockIdx.x * blockDim.x;
    for (int i = tid; i <n; i += blockDim.x * gridDim.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/cuda4dummies/ii/t/stream_test.cu

# 0_Simple/simpleStreams

## CUDA SDK cont.

```
const int N = 1048576;

__global__ void kernel(float *x, int n)
{
    int tid = threadIdx.x + blockIdx.x * blockDim.x;
    for (int i = tid; i <n; i += blockDim.x * gridDim.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/cuda4dummies/ii/t/stream_test.cu

# 0_Simple/simpleStreams

## CUDA SDK cont.

```
const int N = 1048576;

__global__ void kernel(float *x, int n)
{
    int tid = threadIdx.x + blockIdx.x * blockDim.x;
    for (int i = tid; i <n; i += blockDim.x * gridDim.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

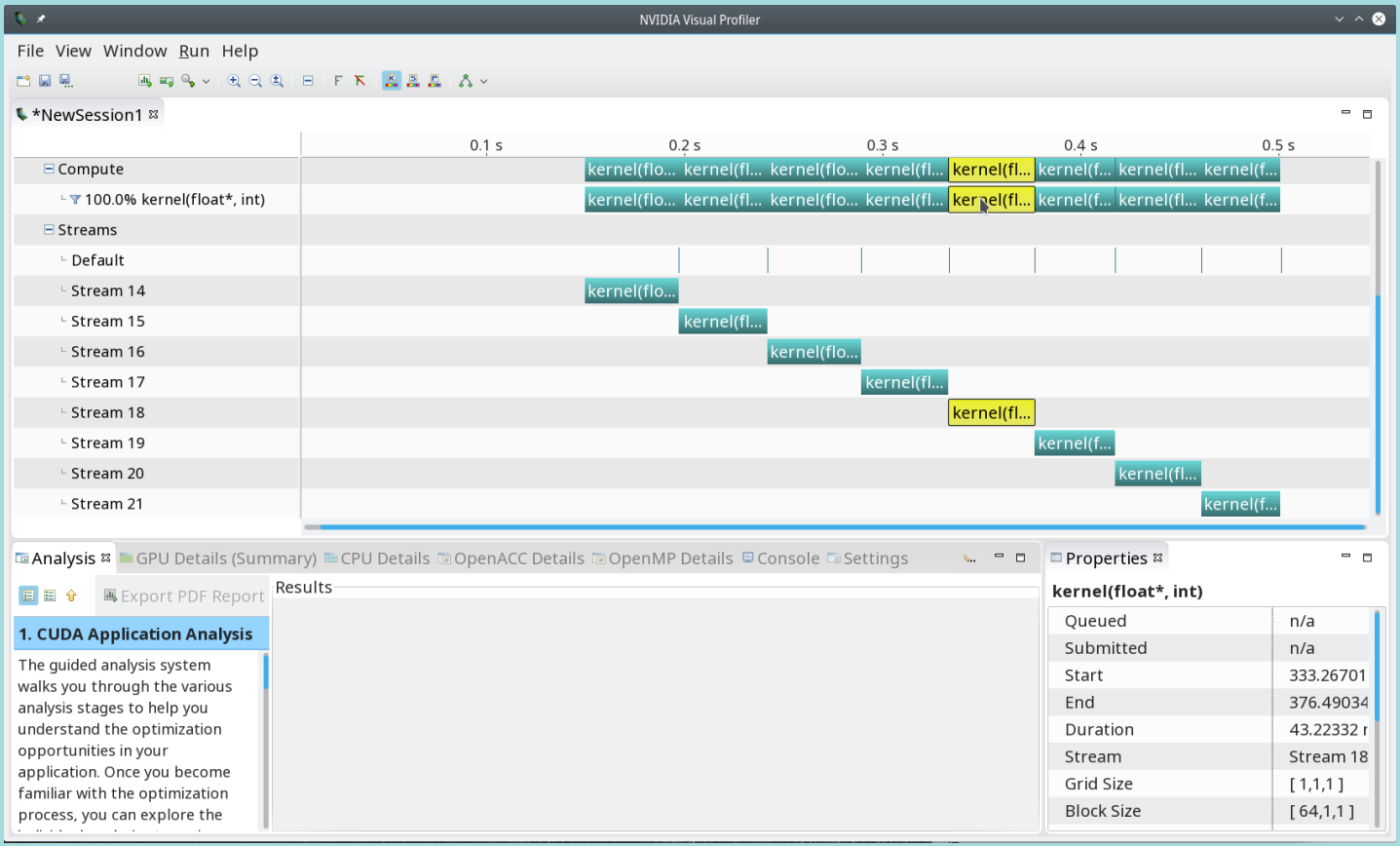
Kernel launch via streams

→ https://tinyurl.com/cuda4dummies/ii/t/stream_test.cu

# 0_Simple/simpleStreams

## CUDA SDK cont.

```
[sh@n71-006]$ nvcc ./stream_test.cu -o ./stream_legacy
[sh@n71-006]$ nvvp ./stream_legacy
```

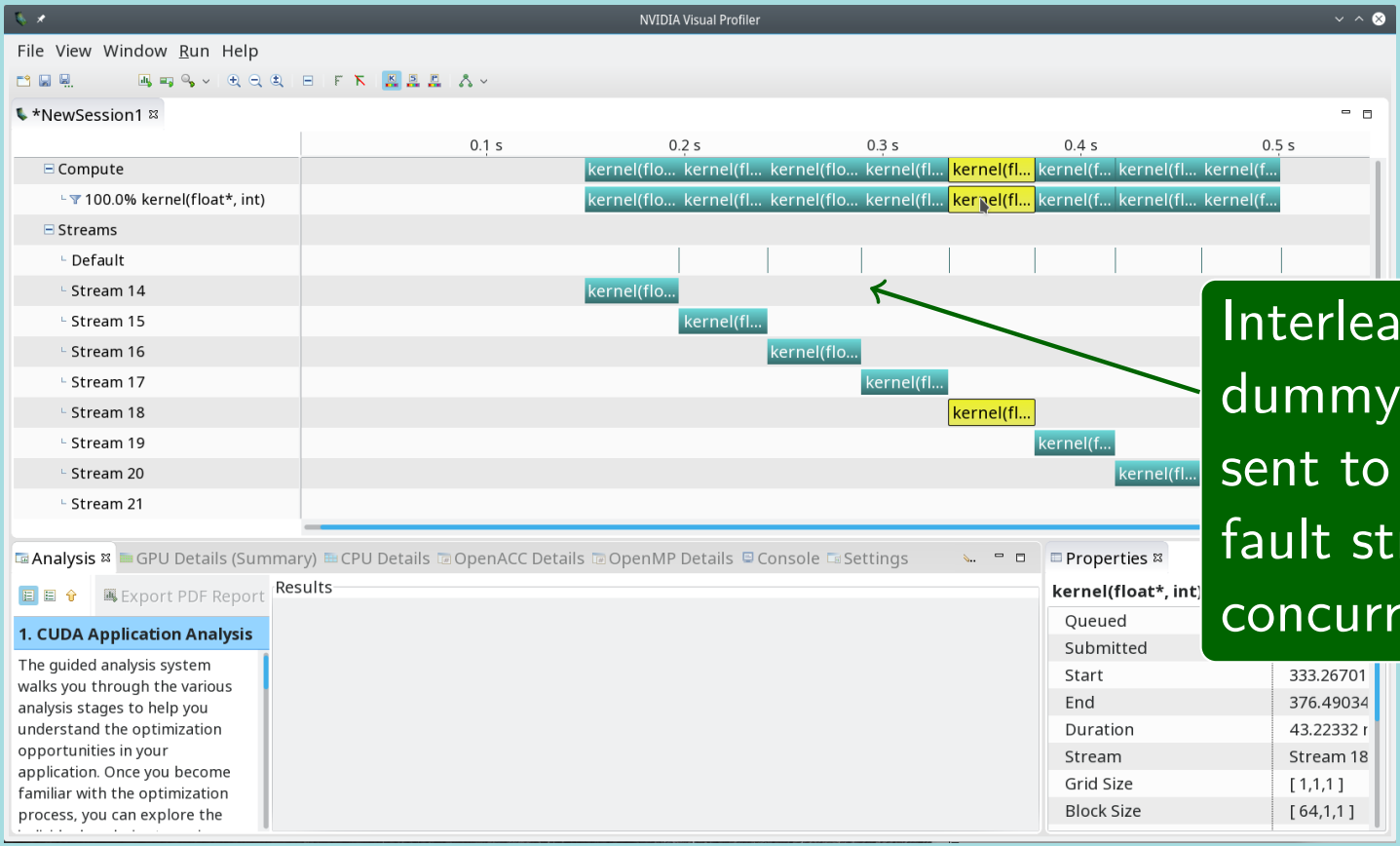



→ https://devblogs.nvidia.com/gpu-pro-tip-cuda-7-streams-simplify-concurrency

# 0_Simple/simpleStreams

## CUDA SDK cont.

```
[sh@n71-006]$ nvcc ./stream_test.cu -o ./stream_legacy
[sh@n71-006]$ nvvp ./stream_legacy
```



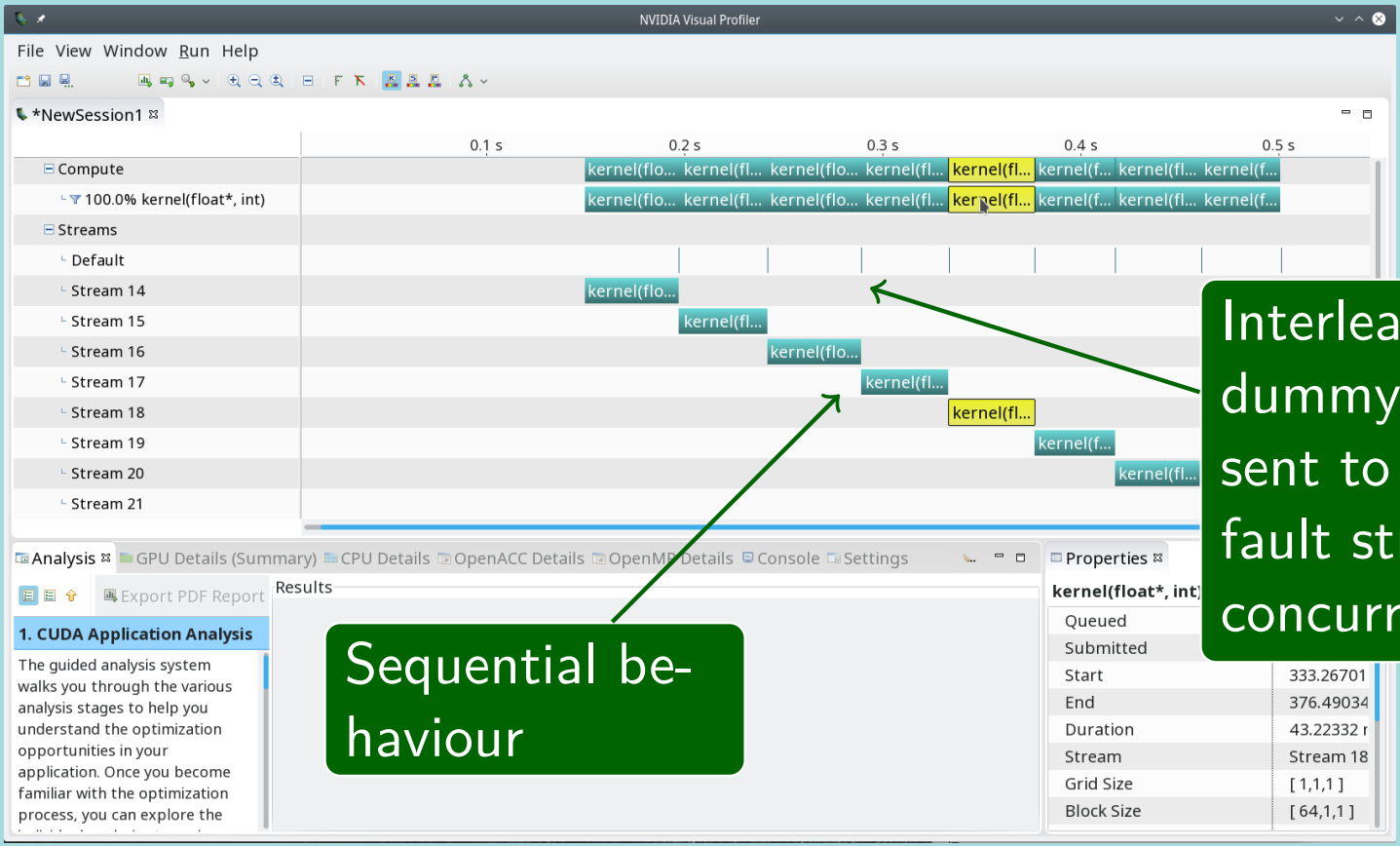

Interleaved dummy kernel sent to the default stream → no concurrency

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

# 0_Simple/simpleStreams

## CUDA SDK cont.

```
[sh@n71-006]$ nvcc ./stream_test.cu -o ./stream_legacy
[sh@n71-006]$ nvvp ./stream_legacy
```

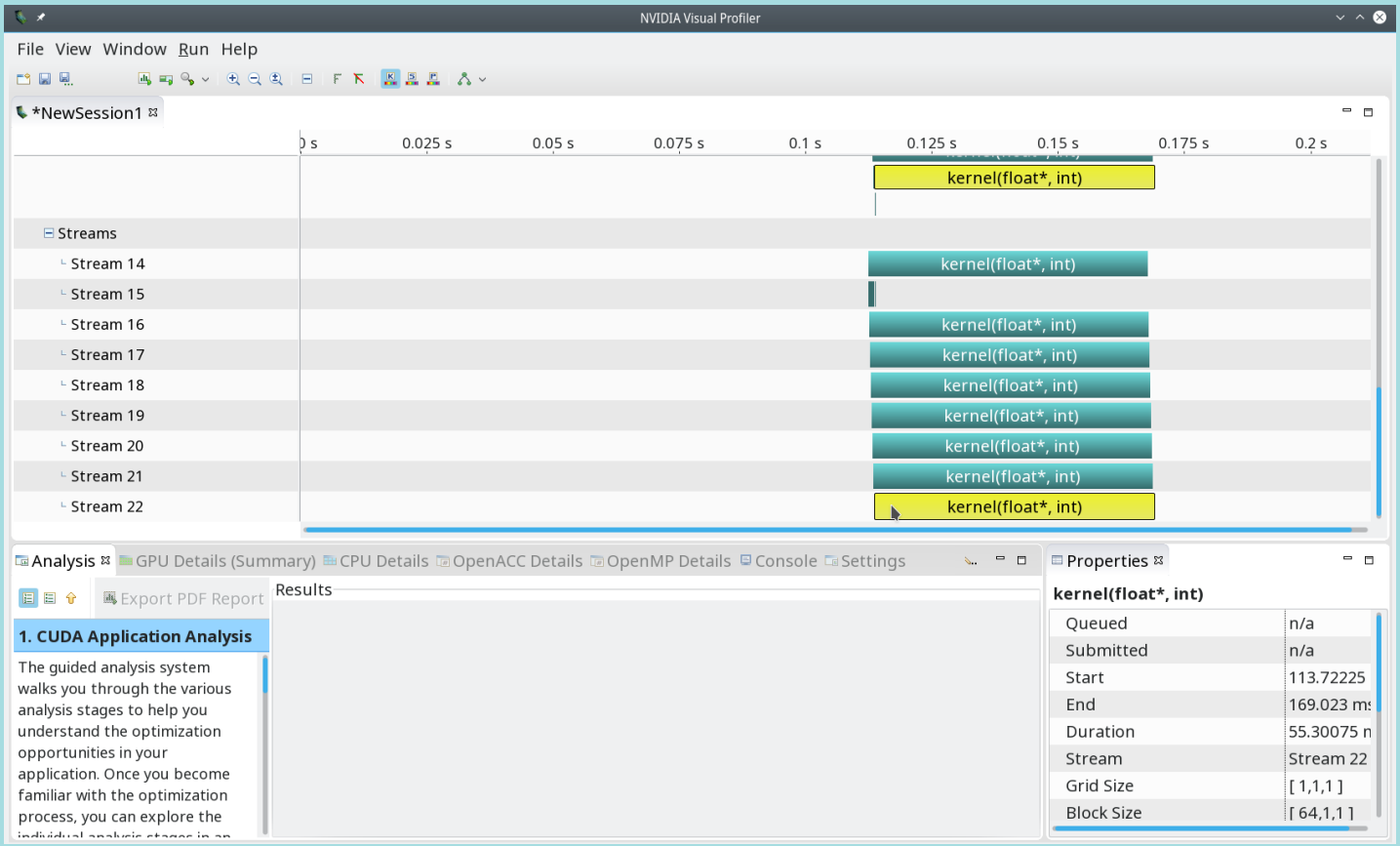



→ https://devblogs.nvidia.com/gpu-pro-tip-cuda-7-streams-simplify-concurrency

# 0_Simple/simpleStreams

## CUDA SDK cont.

```
[sh@n71-006]$ nvcc --default-stream per-thread ./stream_test.cu -o ./stream_per-thread
[sh@n71-006]$ nvvp ./stream_per-thread
```

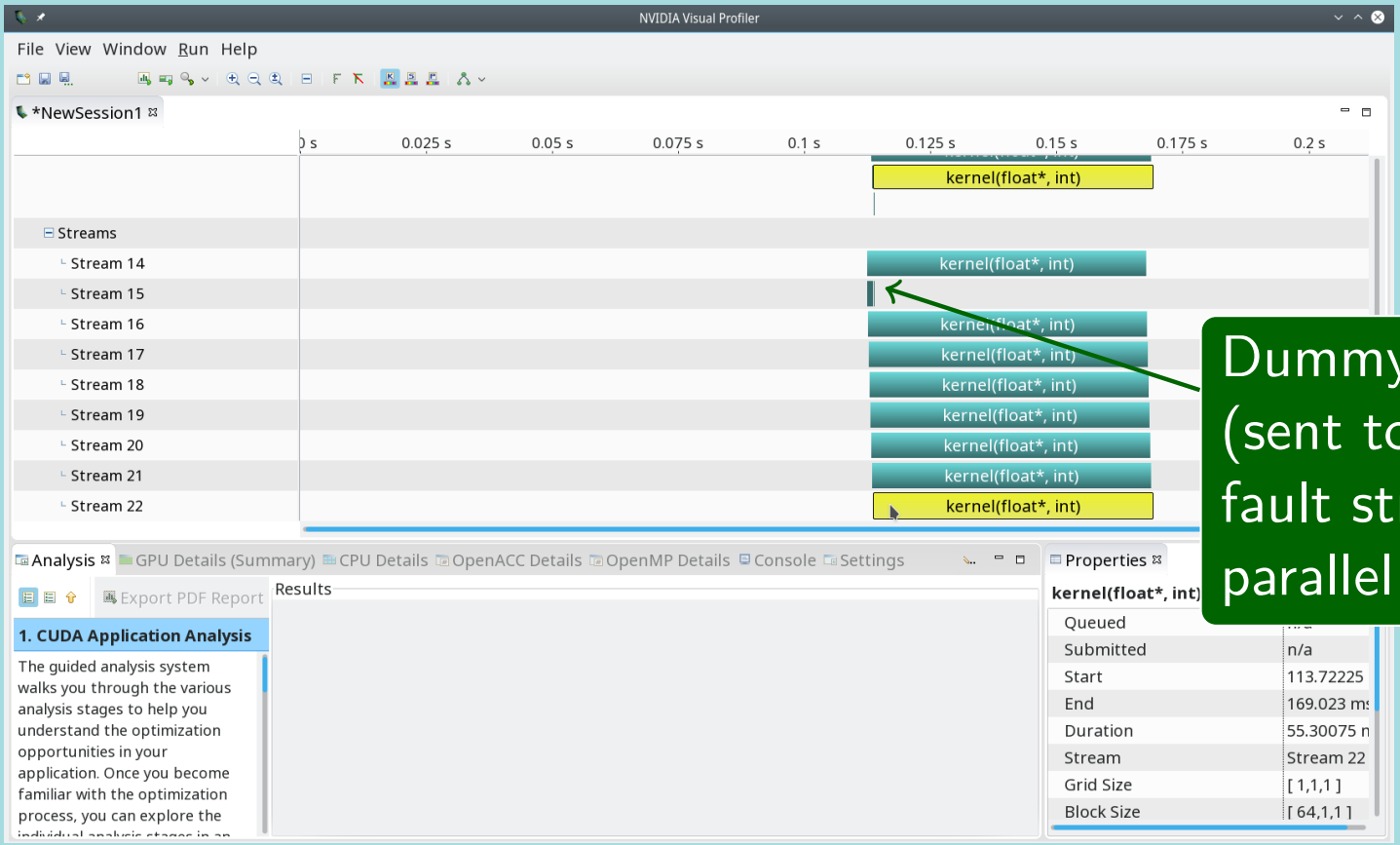



→ https://devblogs.nvidia.com/gpu-pro-tip-cuda-7-streams-simplify-concurrency

# 0_Simple/simpleStreams

## CUDA SDK cont.

```
[sh@n71-006]$ nvcc --default-stream per-thread ./stream_test.cu -o ./stream_per-thread
[sh@n71-006]$ nvvp ./stream_per-thread
```

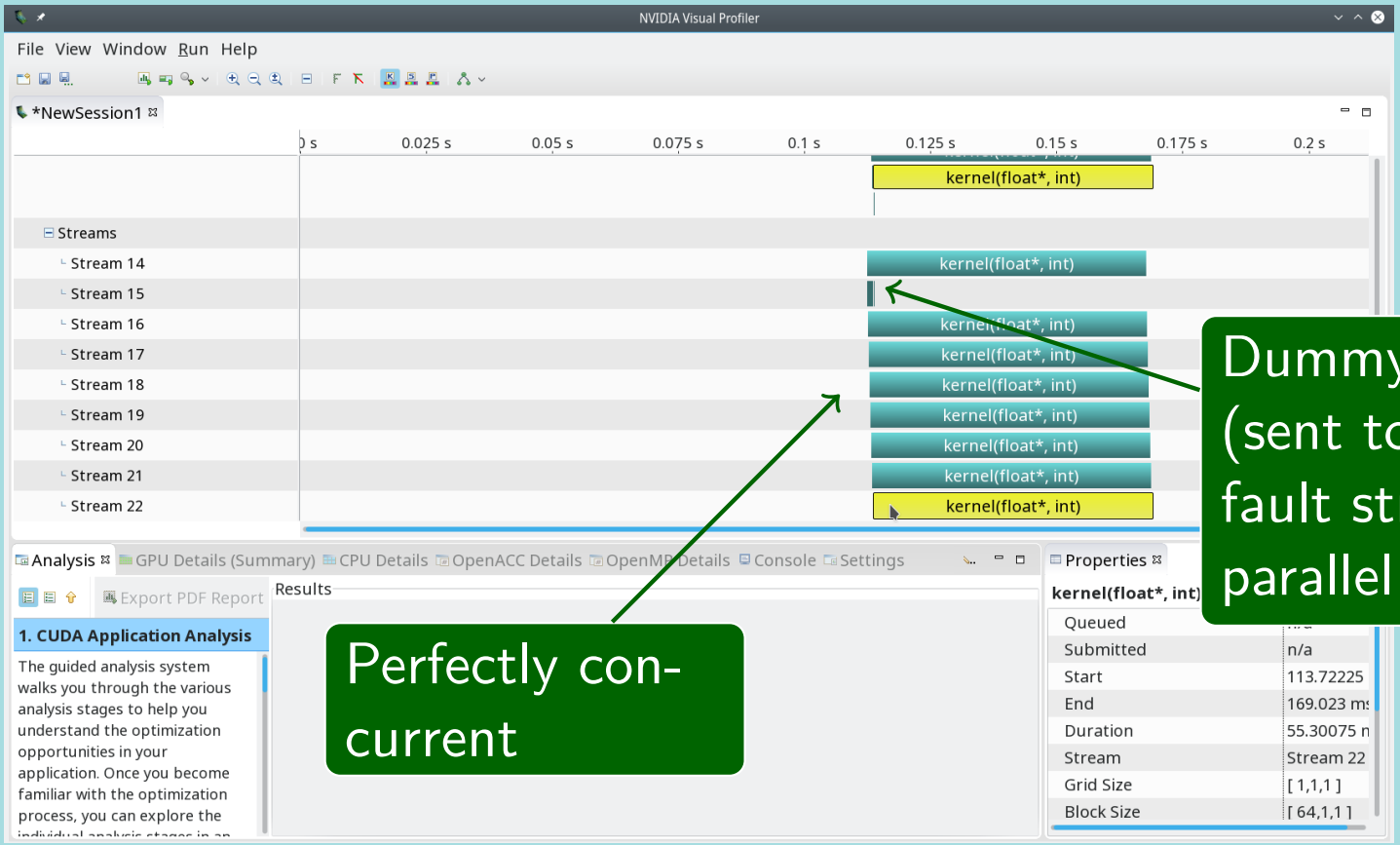



→ https://devblogs.nvidia.com/gpu-pro-tip-cuda-7-streams-simplify-concurrency

# 0_Simple/simpleStreams

## CUDA SDK cont.

```
[sh@n71-006]$ nvcc --default-stream per-thread ./stream_test.cu -o ./stream_per-thread
[sh@n71-006]$ nvvp ./stream_per-thread
```




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

# 0_Simple/cudaOpenMP (variant)

## CUDA SDK cont.

```
#include <omp.h>
const int N = 1048576;
__global__ void kernel(float *x, int n)
{
    int tid = threadIdx.x + blockIdx.x * blockDim.x;
    for (int i = tid; i <n; i += blockDim.x * gridDim.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://devblogs.nvidia.com/gpu-pro-tip-cuda-7-streams-simplify-concurrency

# 0_Simple/cudaOpenMP (variant)

## CUDA SDK cont.

```
#include <omp.h>
const int N = 1048576;
__global__ void kernel(float *x, int n)
{
    int tid = threadIdx.x + blockIdx.x * blockDim.x;
    for (int i = tid; i <n; i += blockDim.x * gridDim.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

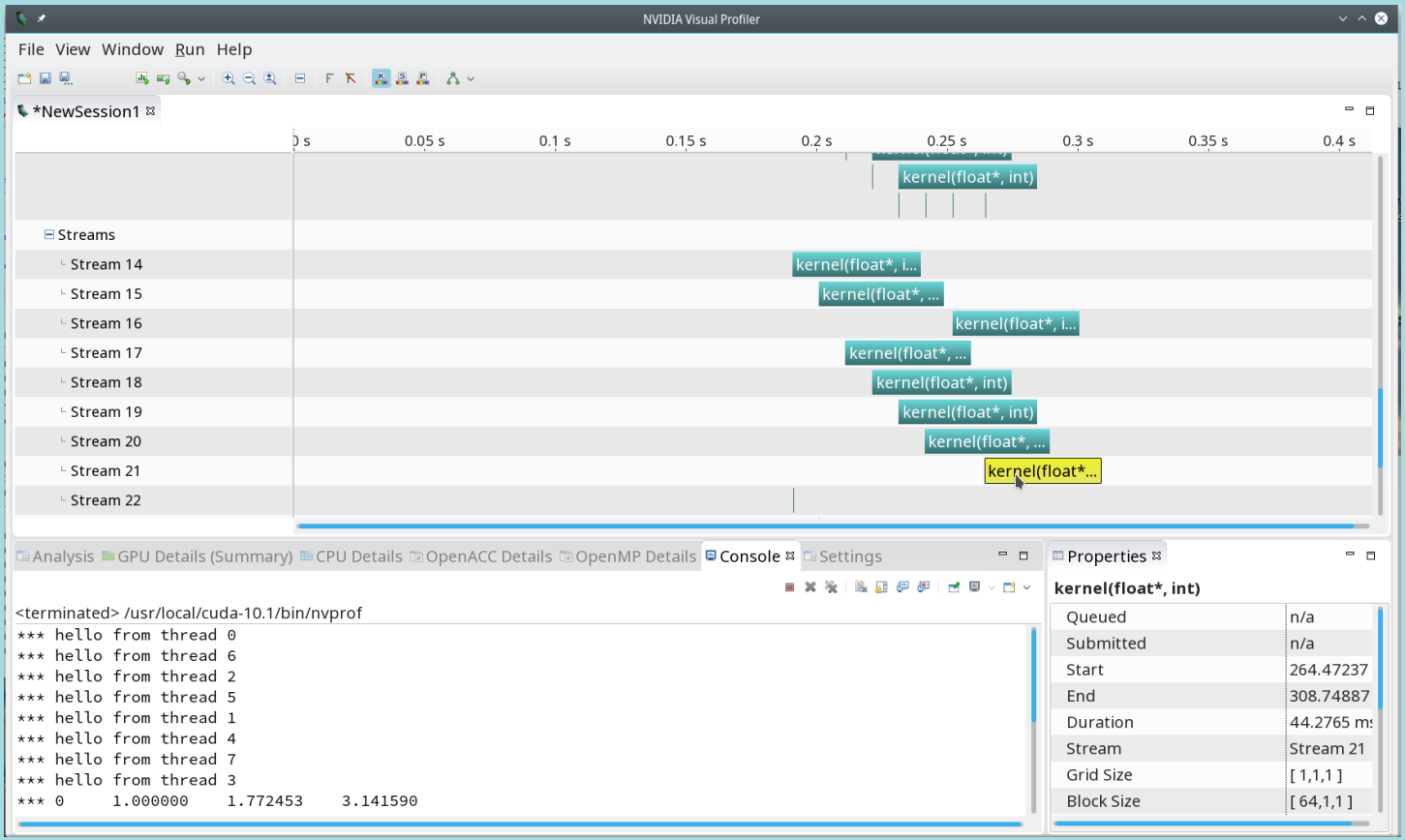
Simplest way of exhausting all available CPU/GPU resources

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

# 0_Simple/cudaOpenMP (variant)

## CUDA SDK cont.

```
[sh@n71-006]$ nvcc --default-stream per-thread -ccbin g++ -m64 -Xcompiler -fopenmp \
./stream_test_v5.cu -o ./stream_v5_per-thread
[sh@n71-006]$ nvvp ./stream_v5_per-thread
```

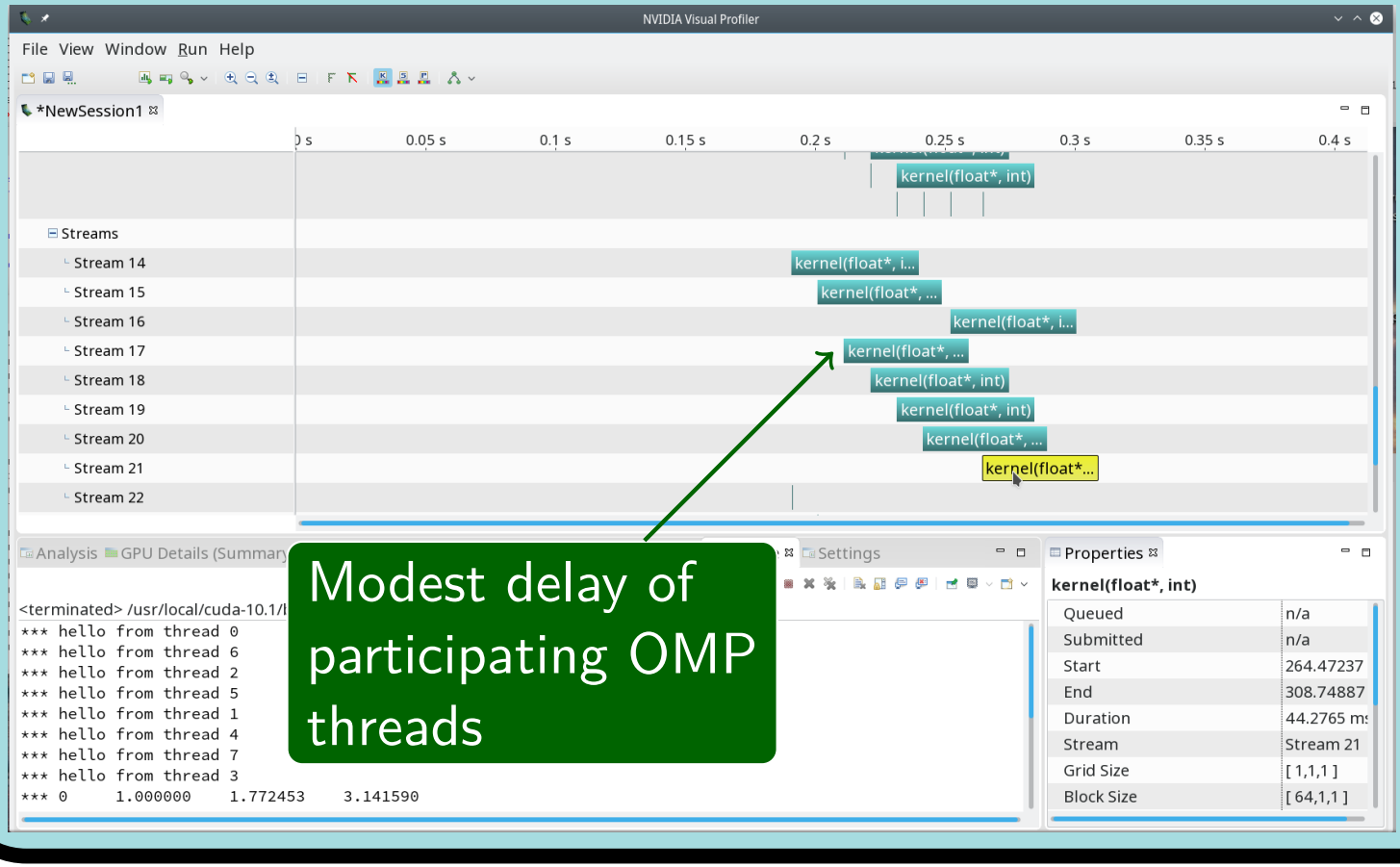



→ https://devblogs.nvidia.com/gpu-pro-tip-cuda-7-streams-simplify-concurrency

# 0_Simple/cudaOpenMP (variant)

## CUDA SDK cont.

```
[sh@n71-006]$ nvcc --default-stream per-thread -ccbin g++ -m64 -Xcompiler -fopenmp \
./stream_test_v5.cu -o ./stream_v5_per-thread
[sh@n71-006]$ nvvp ./stream_v5_per-thread
```

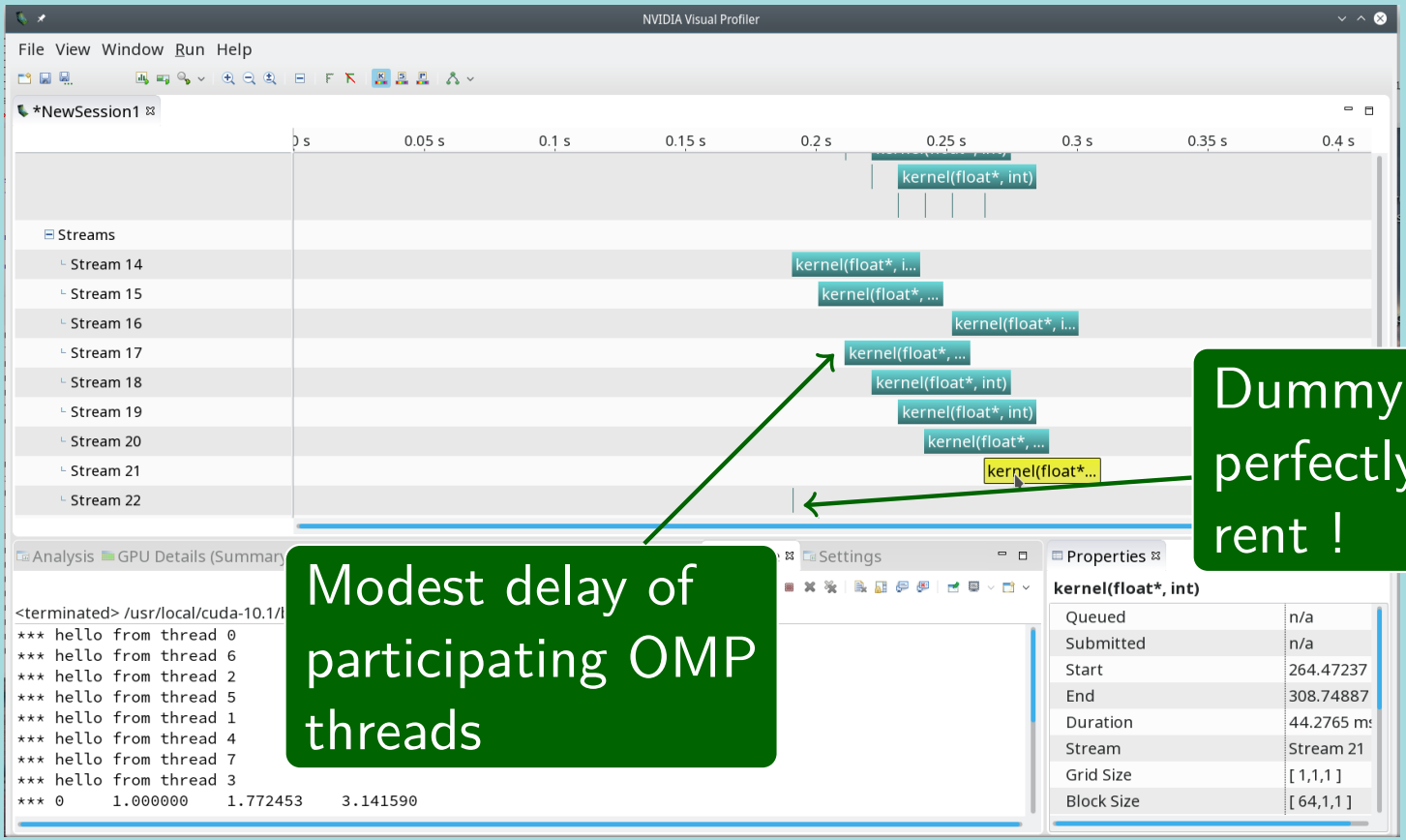



→ https://devblogs.nvidia.com/gpu-pro-tip-cuda-7-streams-simplify-concurrency

# 0_Simple/cudaOpenMP (variant)

## CUDA SDK cont.

```
[sh@n71-006]$ nvcc --default-stream per-thread -ccbin g++ -m64 -Xcompiler -fopenmp \
./stream_test_v5.cu -o ./stream_v5_per-thread
[sh@n71-006]$ nvvp ./stream_v5_per-thread
```




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

# 0_Simple/cudaOpenMP (variant)

## CUDA SDK cont.

- 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()
- nvvp is a nice tool too !

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

# 7_CUDALibraries/randomFog

## CUDA SDK cont.

- CUDA is for graphics cards, so there are a lot of graphics examples too
- Example 7_CUDALibraries/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

→ `/usr/local/cuda/extras/demo_suite/randomFog`

# 5_Simulations/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

→ `/usr/local/cuda/extras/demo_suite/nbody`

# 5_Simulations/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

→ `/usr/local/cuda/extras/demo_suite/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 categories, albeit still predominantly an exploratory space (C/C++, error-handling, memory management...)

# Take Home Messages

- CUDA SDK — a rich playground for beginners interested in learning the basics of GPU computing
- Organized into different categories, albeit still predominantly an exploratory space (C/C++, error-handling, memory management...)
- PCIe bandwidth remains a central limitation of GPU computing

# Take Home Messages

- CUDA SDK — a rich playground for beginners interested in learning the basics of GPU computing
- Organized into different categories, albeit still predominantly an exploratory space (C/C++, error-handling, memory management...)
- PCIe bandwidth remains a central limitation of GPU computing
- CUDA streams bring in another level of flexibility, especially when run concurrently (perhaps from individual OMP 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 categories, albeit still predominantly an exploratory space (C/C++, error-handling, memory management...)
- PCIe bandwidth remains a central limitation of GPU computing
- CUDA streams bring in another level of flexibility, especially when run concurrently (perhaps from individual OMP threads on the host)
- nvvp is another very helpful tool

# Take Home Messages

- CUDA SDK — a rich playground for beginners interested in learning the basics of GPU computing
- Organized into different categories, albeit still predominantly an exploratory space (C/C++, error-handling, memory management...)
- PCIe bandwidth remains a central limitation of GPU computing
- CUDA streams bring in another level of flexibility, especially when run concurrently (perhaps from individual OMP threads on the host)
- nvvp is another very helpful tool
- Graphical demos — nice to have them too !