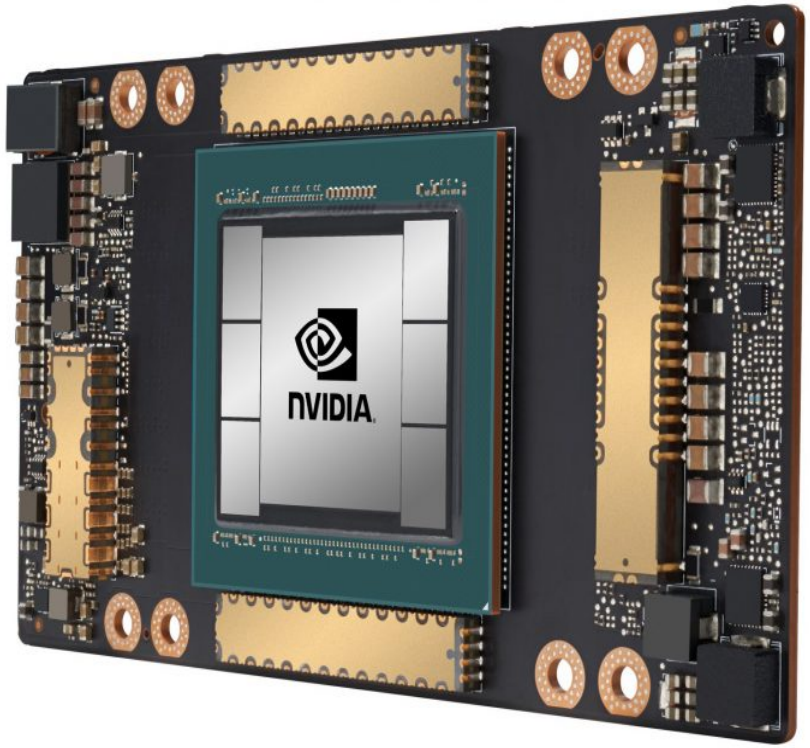
Profiling GPU codes with Nsight

Sergey Mashchenko
syam@sharcnet.ca
SHARCNET / Compute Ontario / Alliance

May 18, 2022

Overview

- Why?
- Where?
- How?
- Demo



Why to profile GPU codes?

- GPUs are significantly more expensive, and less available, than CPUs
- Not all research codes / algorithms are well suited for GPU acceleration.
- As a consequence, profiling is a critical step in a GPU code development, and should start from the very first kernel you write.

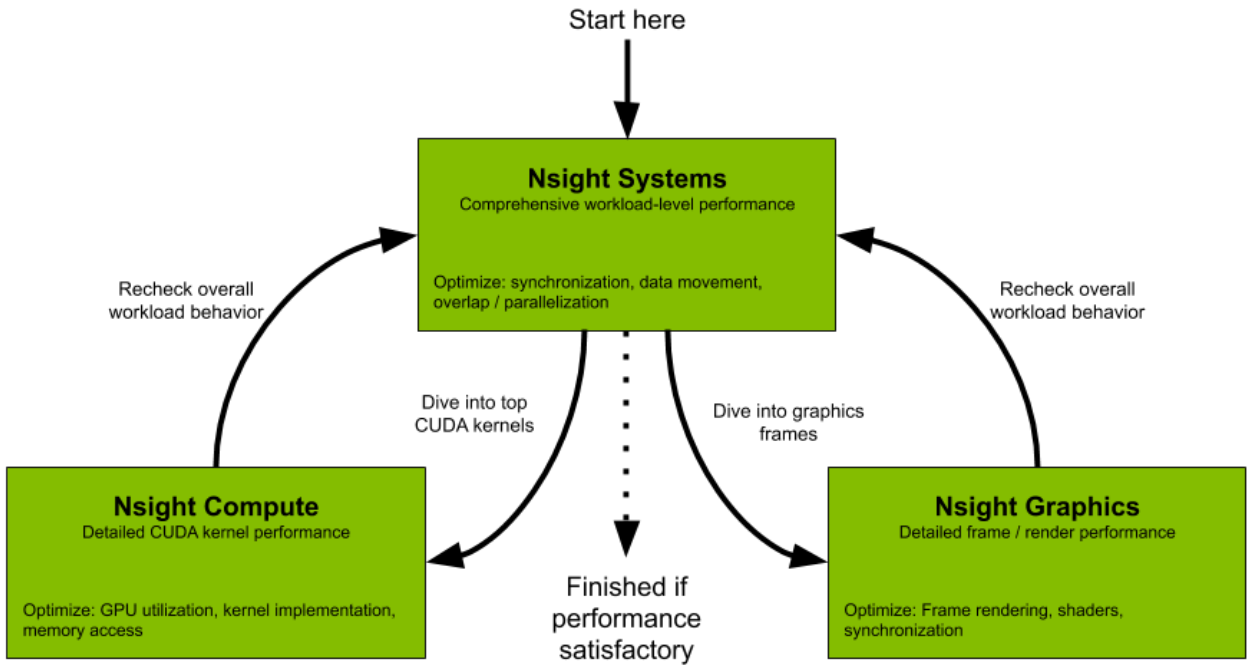
Tools

- NVIDIA is has produced an retired quite a few GPU profilers.
- The following profilers are no longer maintained (though still available, work up to V100):
  - **`nvprof`**: command-line function-level profiler, for both GPU and CPU parts of the code
  - **`nvvp`**: graphical (GUI) profiler

Tools: Nsight

- NVIDIA also maintained for years their Nsight suite of products (IDE / debugger / profiler).
  - Nsight Eclipse edition (Linux, Mac)
  - Nsight Visual Studio Edition (Windows)
- Since 2018, Nsight profilers became also available as three stand alone packages: Nsight Compute, Nsight Systems, and Nsight Graphics.

Nsight packages



Where to run Nsight

- If you have a fairly capable recent GPU inside your laptop / PC, you can install and use Nsight suite (Eclipse for Linux/Mac, Visual Studio for Windows) on your own computer.
- But if the goal is to optimize your code for Alliance national systems, you definitely want to run Nsight remotely on our clusters.
  - Command line (CLI) tools can be submitted as jobs
  - Interactive GUI tools can be used on compute nodes allocated with `salloc`, with either X11 or VNC connections (more about that later).
- If your internet is too slow, you can run CLI Nsight tools on a cluster, then analyze the results on your computer using GUI Nsight.

How to use GUI tools on clusters

- X11 forwarding (MobaXterm for Windows, Xquartz for Mac):

```
$ ssh -Y graham.computecanada.ca
$ salloc --x11 …
```

- VNC on a compute node (requires two terminal windows)

```
[login_node]$ salloc …
[gra123]$ export XDG_RUNTIME_DIR=${SLURM_TMPDIR}
[gra123]$ vncserver
[your_PC]$ ssh graham.computecanada.ca -NL 5902:gra123:5901
```

  - On your PC, launch TigerVNC viewer
  - Enter the destination: localhost:5902

VNC helper script

```
#!/bin/bash

# Required for vncserver:
export XDG_RUNTIME_DIR=${SLURM_TMPDIR}

# Starting VNC server, recording the channel:
N=$(vncserver 2>&1 |grep "^New" |cut -d: -f3)

# Computing the remote port:
Rport=$((5900 + $N))

# Printing the command for the local computer:
echo "ssh $USER@$SLURM_CLUSTER_NAME.computecanada.ca -NL 5902:$SLURMD_NODENAME:$Rport"
```

NVIDIA profilers on our clusters

- Loaded when a cuda module is loaded, e.g.
  `$ module load cuda/11.4`
- Old tools (up to Volta: P100, V100):
  - `nvprof` (CLI)
  - `nvvp` (GUI)
- Nsight tools (work on P100*, V100, T4, A100):
  - `ncu, ncu-ui`: Nsight Compute (CLI / GUI; V100 and up)
  - `nsys, nsys-ui`: Nsight Systems (CLI / GUI; P100 and up)

Compiling code

- Compile the code as usual (after loading the cuda module)
  - The only extra compiler switch required is `-lineinfo` (do not use `-G` – that one is for debugging)

```
$ module load cuda
$ nvcc -O2 -arch=sm_70 -lineinfo code.cu
```

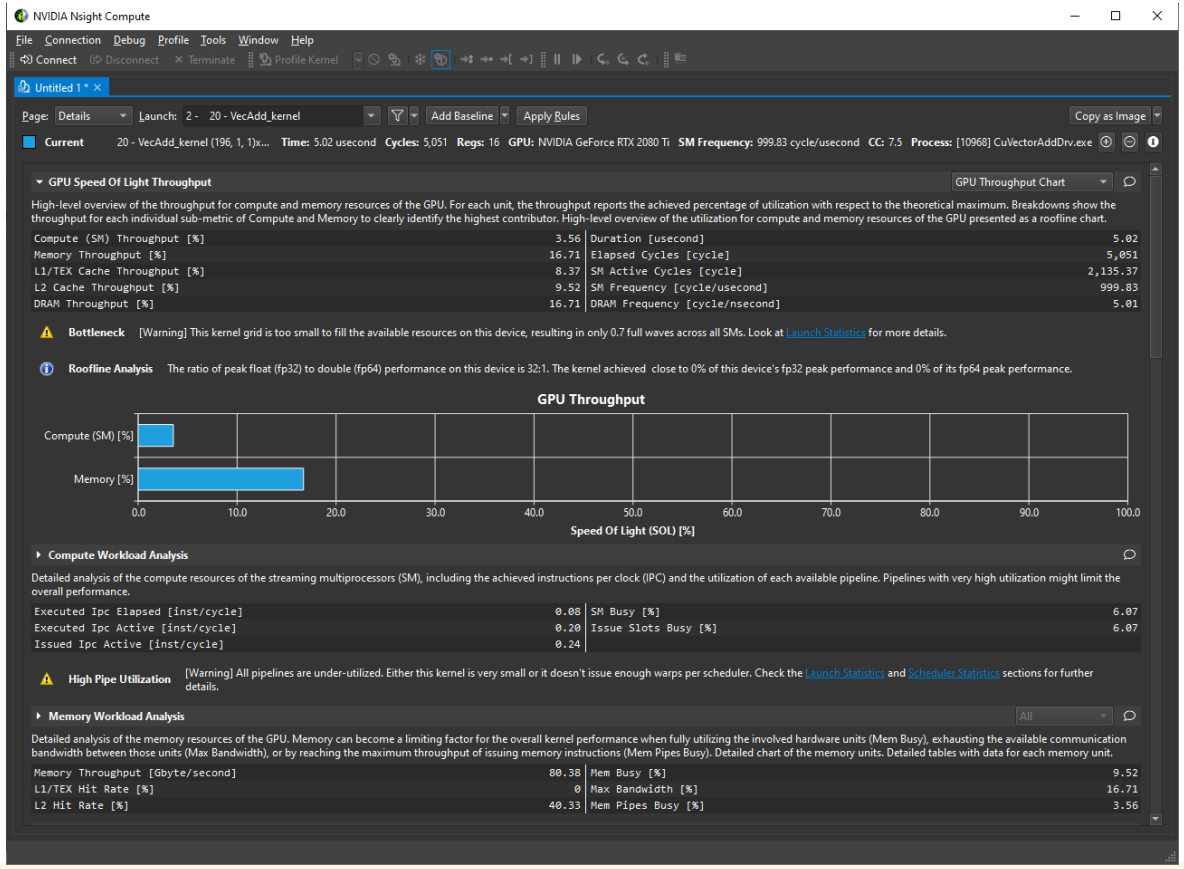
Nsight Compute

- Command: ncu, ncu-ui

```
$ ncu -o output_file code
```

- Typically the first step in profiling a GPU code
- Allows one to maximize the performance of individual kernels
- Full documentation: https://docs.nvidia.com/nsight-compute/NsightCompute

Kernel metrics



Line-by-line info

View: Source and SASS

Source: matrixMul.cu | Find...

Navigation: Instructions Executed

#	Source	Sampling Data (All)	Instructions Executed	Instructions Executed-On Thread
87	As[ty][tx] = A[a + wA * ty + tx];	1,230	3,712,000	118,784,000
88	Bs[ty][tx] = B[b + wB * ty + tx];	1,259	3,712,000	118,784,000
89		0		
90	// Synchronize to make sure the matrice	0		
91	__syncthreads();	41	128,000	4,096,000
92		0		
93	// Multiply the two matrices together;	0		
94	// each thread computes one element	0		
95	// of the block sub-matrix	0		
96	#pragma unroll	0		
97		0		
98	for (int k = 0; k < BLOCK_SIZE; ++k) {	7,542	31,616,000	946,176,000
99	Csub += As[ty][k] * Bs[k][tx];	39,288	161,792,000	5,177,344,000
100	}	0		
101		0		
102	// Synchronize to make sure that the pr	0		
103	// computation is done before loading t	0		
104	// sub-matrices of A and B in the next	0		
105	__syncthreads();	23	128,000	4,096,000
106	}	0		
107		0		
108	// Write the block sub-matrix to device mem	0		
109	// each thread writes one element	0		
110	int c = wB * BLOCK_SIZE * by + BLOCK_SIZE *	2	32,000	1,024,000
111	C[c + wB * ty + tx] = Csub;	31	128,000	4,096,000
112	}	9	12,800	409,600
113		0		

Source: MatrixMulCUDA | Find...

Navigation: Sampling Data (All)

#	Address	Source	Sampling Data (All)	Instructions Executed
209	0000000b 00b9be00	MOV R20, R20	14	64,000
210	0000000b 00b9be10	MOV R23, R20	18	64,000
211	0000000b 00b9be20	MOV R21, R22	21	64,000
212	0000000b 00b9be30	MOV R21, R21	14	64,000
213	0000000b 00b9be40	MOV R22, R21	359	2,112,000
214	0000000b 00b9be50	MOV R20, R23	500	2,112,000
215	0000000b 00b9be60	MOV R22, R22	513	2,112,000
216	0000000b 00b9be70	MOV R20, R20	501	2,112,000
217	0000000b 00b9be80	ISETP.LT.AND P0, PT, R22, 0x20, PT	495	2,112,000
218	0000000b 00b9be90	PLOP3.LUT P0, PT, P0, PT, PT, 0x8, 0	495	2,112,000
219	0000000b 00b9bea0	@P0 BRA 0xb00b9c430	502	2,112,000
220	0000000b 00b9beb0	BRA 0xb00b9bec0	480	2,048,000
221	0000000b 00b9bec0	MOV R21, 0x0	505	2,048,000
222	0000000b 00b9bed0	MOV R21, R21	504	2,048,000
223	0000000b 00b9bee0	MOV R21, R21	522	2,048,000
224	0000000b 00b9bef0	MOV R21, R21	490	2,048,000
225	0000000b 00b9bf00	MOV R23, RZ	514	2,048,000
226	0000000b 00b9bf10	MOV R26, c[0x0][0x18]	481	2,048,000
227	0000000b 00b9bf20	MOV R27, c[0x0][0x1c]	494	2,048,000
228	0000000b 00b9bf30	IADD3 R26, P0, R21, R26, RZ	470	2,048,000
229	0000000b 00b9bf40	IADD3.X R27, R23, R27, RZ, P0, !PT	469	2,048,000
230	0000000b 00b9bf50	MOV R24, R12	517	2,048,000
231	0000000b 00b9bf60	SHF.R.S32.HI R25, RZ, 0x1f, R24	469	2,048,000
232	0000000b 00b9bf70	MOV R24, R24	488	2,048,000
233	0000000b 00b9bf80	MOV R25, R25	490	2,048,000
234	0000000b 00b9bf90	MOV R21, R24	457	2,048,000
235	0000000b 00b9bfa0	MOV R23, R25	467	2,048,000

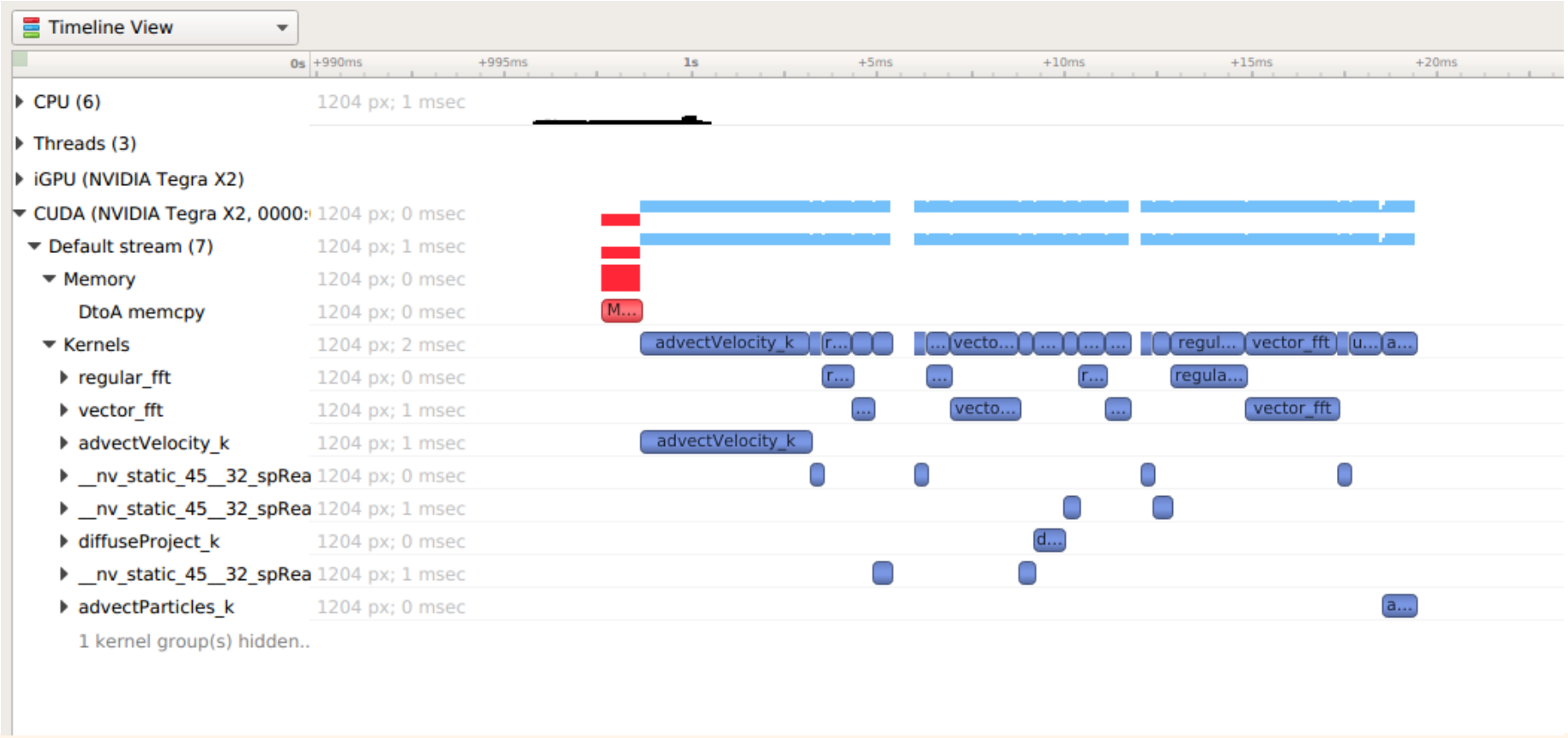
Nsight Systems

- Command: nsys, nsys-ui

```
$ nsys profile -o output_file code
```

- Typically the last step in profiling a GPU code
- Fine-tuning the interactions between kernels, memcopies, CPU code etc, both synchronous and asynchronous
- Full documentation: https://docs.nvidia.com/nsight-systems/UserGuide

CUDA trace

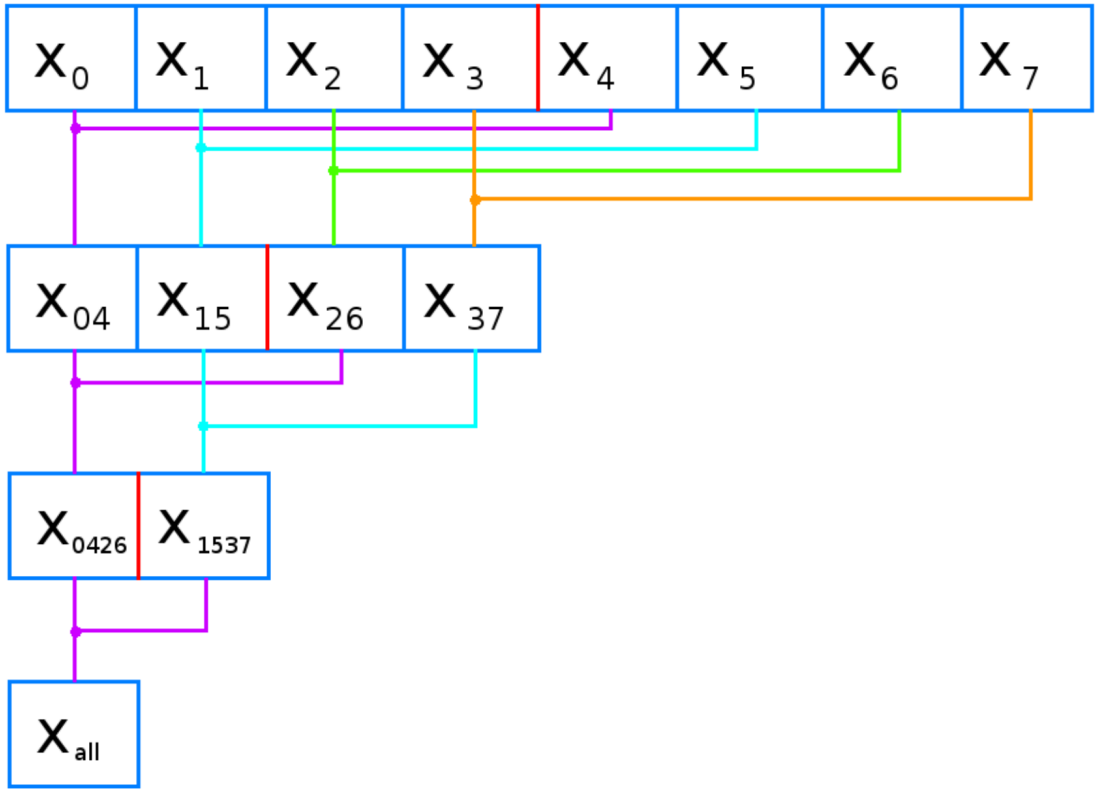

Timeline View
0s
+990ms
+995ms
1s
+5ms
+10ms
+15ms
+20ms
CPU (6)
1204 px; 1 msec
Threads (3)
iGPU (NVIDIA Tegra X2)
CUDA (NVIDIA Tegra X2, 0000:
1204 px; 0 msec
Default stream (7)
1204 px; 1 msec
Memory
1204 px; 0 msec
DtoA memcpy
1204 px; 0 msec
Kernels
1204 px; 2 msec
regular_fft
1204 px; 0 msec
vector_fft
1204 px; 1 msec
advectVelocity_k
1204 px; 1 msec
__nv_static_45__32_spRea
1204 px; 0 msec
__nv_static_45__32_spRea
1204 px; 1 msec
diffuseProject_k
1204 px; 0 msec
__nv_static_45__32_spRea
1204 px; 1 msec
advectParticles_k
1204 px; 0 msec
1 kernel group(s) hidden..
M...
advectVelocity_k
r...
...
vecto...
regul...
vector_fft
u...
a...
regula...
d...

Live demo

Reduction code

- The code adds up elements of a long vector on GPU.
- The first version uses a very slow method: serialized summation using `atomicAdd()` function.
- The second version uses the proper way: parallel summation using binary reduction.

Binary reduction



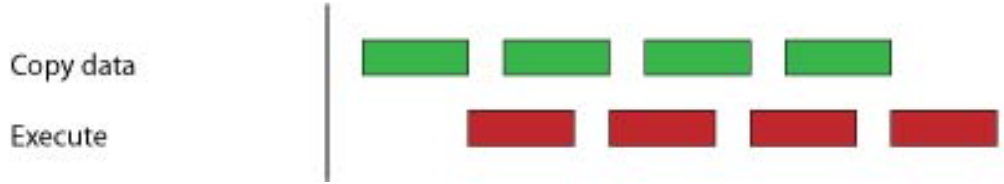
Staged copy/compute code

- The first version of the code first copies a long vector to GPU, then carries out independent per-element computations.
- The second version hides some of the costs of copying data to GPU by running copying and computing in parallel, using two GPU streams.

Staged copy/compute algorithm

One stream scenario:

You need two streams for this:



Thank you!