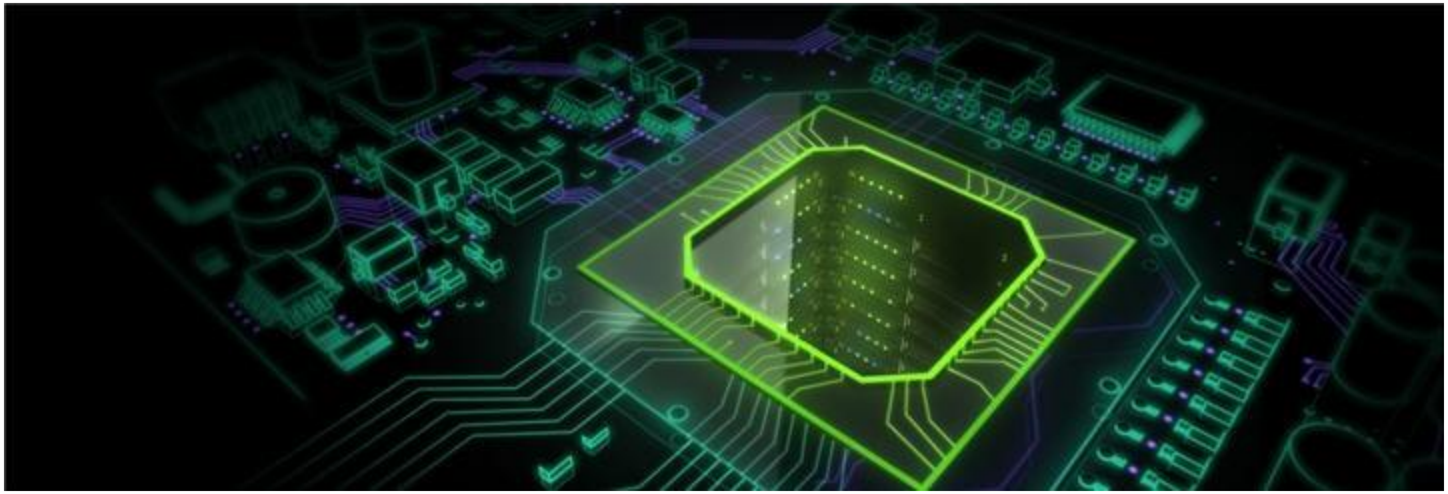
CSCI-GA.3033-004

Graphics Processing Units (GPUs): Architecture and Programming

Tools

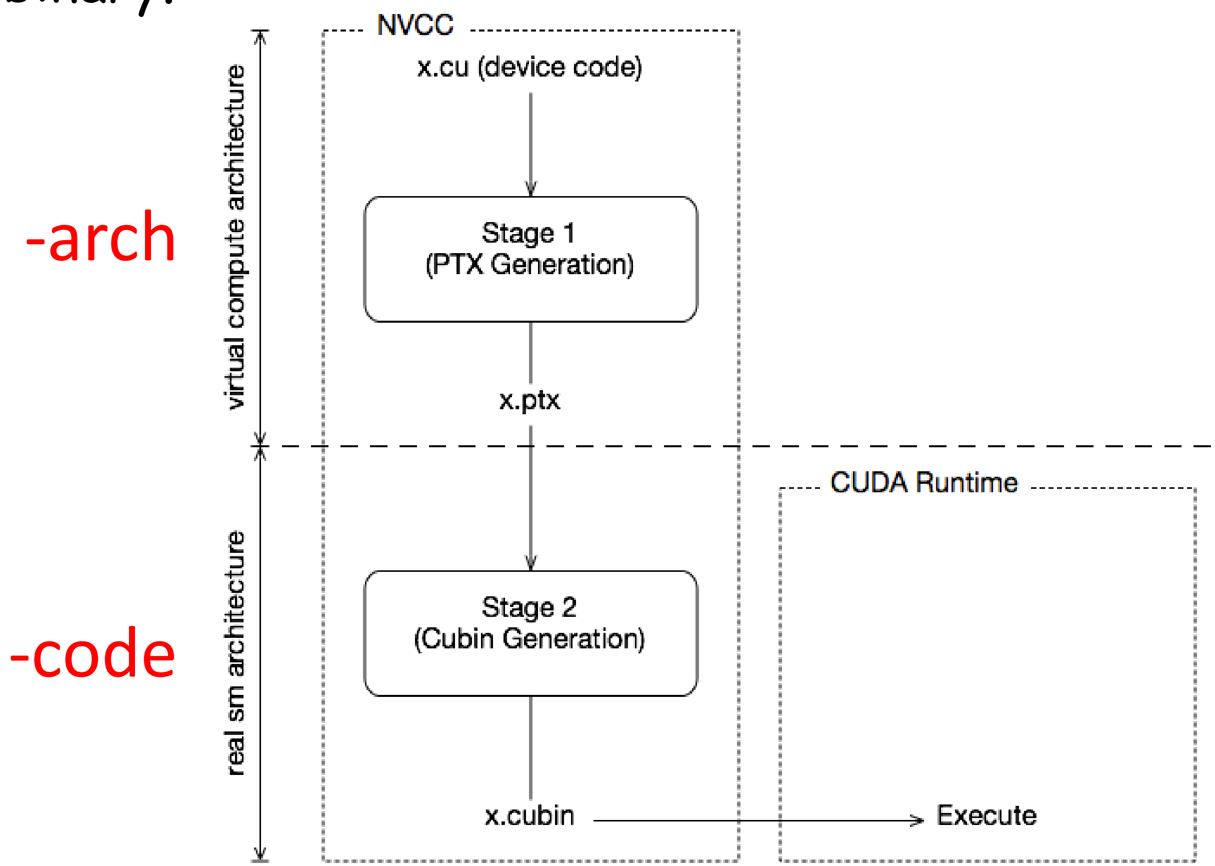
Mohamed Zahran (aka Z)
mzahran@cs.nyu.edu
http://www.mzahran.com



Compilation: nvcc

NVCC device specific switches

- `-arch` : controls the "virtual'' architecture that will be used for the generation of the PTX code.
- `-code` : specifies the actual device that will be targeted by the cubin binary.

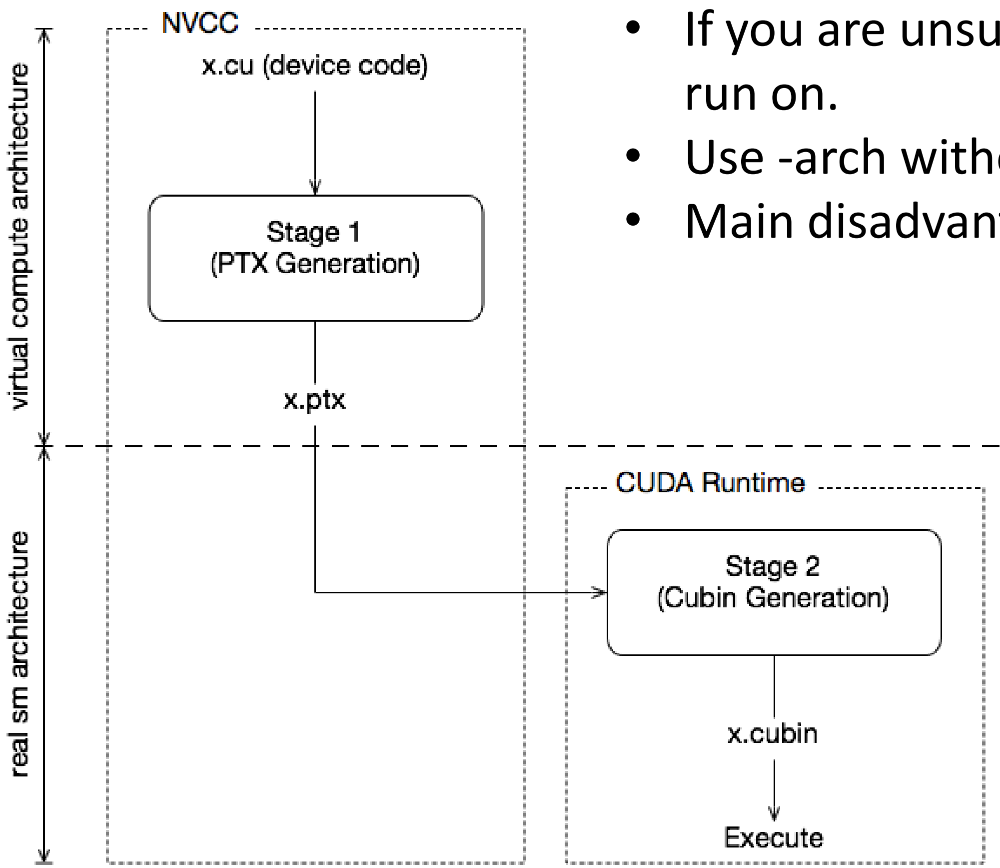


sm_xy

- x is the GPU generation number
- y is the version within that generation
- Binary compatibility of GPU applications is not guaranteed across different generations.
  - Example: a CUDA application that has been compiled for a Fermi GPU will very likely not run on a Kepler GPU (and vice versa).
- This is why nvcc relies on a two stage compilation model for ensuring application compatibility with future GPU generations.

-arch	-code	
compute_30 compute_32	sm_30 sm_32	Basic features + Kepler + unified memory
compute_35	sm_35	+ dynamic parallelism
compute_50 compute_52 compute_53	sm_50 sm_52 sm_53	+ Maxwell support
compute_60 compute_61 compute_62	sm_60 sm_61 sm_62	+ Pascal support
compute_70 compute_72	sm_70 sm_72	+ Volta support
compute_75	sm_75	+ Turing support

JIT Compilation



- If you are unsure which exact GPU the code will run on.
- Use -arch without -code
- Main disadvantage: slower startup

Fatbinaries

nvcc x.cu -arch=compute_30 -code=compute_30,sm_30,sm_35

generate PTX
and keep it in the
binary generated,
for JIT on future GPUs

Generate binaries for
two versions of Kepler

At runtime, the CUDA driver will select the most appropriate translation when the device function is launched.

Till now we have single virtual architecture and several real architectures.
How about several virtual architectures?

--generate-code (or -gencode)

```
nvcc x.cu \
--generate-code arch=compute_20,code=sm_20 \
--generate-code arch=compute_20,code=sm_21 \
--generate-code arch=compute_30,code=sm_30
```

The Default

nvcc x.cu

is equivalent to

nvcc x.cu -arch=compute_20 -code=sm_20,compute_20

nvcc

- Some nvcc features: --ptxas-options=-v
  - Print the smem, register and other resource usages
- Generates CUDA binary file: nvcc –cubin
  - cubin file is the cuda executable

Deciphering ptxas -v

gmem: Global memory

smem CUDA shared memory

lmem CUDA local memory (thread-private, spilled)

cmem constant memory

- cmem[0] kernel arguments
- cmem[1] variables
- cmem[2] user defined constant objects
- cmem[14] compiler generated constants
- cmem[16] compiler generated constants

Profiling: nvprof

nvprof

- CUDA profiler: profiling data from the command line

```
$ nvprof [nvprof_args] <app> [app_args]
```

- To profile a region of the application:
  1. #include <cuda_profiler_api.h>
  2. in the host function surround the region with:
     - cudaProfilerStart()
     - cudaProfilerStop()
  3. nvcc myprog.cu
  4. nvprof --profile-from-start-off ./a.out

nvprof summary mode (default)

command line: nvprof progname [prog args]

==44825== NVPROF is profiling process 44825, command: ./vecadd 100000000 1 1024 32
==44825== Profiling application: ./vecadd 100000000 1 1024 32
==44825== Profiling result:

Time(%)	Time	Calls	Avg	Min	Max	Name
56.07%	385.56ms	2	192.78ms	113.38ms	272.18ms	[CUDA memcpy HtoD]
26.47%	181.99ms	1	181.99ms	181.99ms	181.99ms	addvector(int*, int*, int*$
17.46%	120.07ms	1	120.07ms	120.07ms	120.07ms	[CUDA memcpy DtoH]

==44825== API calls:

Time(%)	Time	Calls	Avg	Min	Max	Name
73.22%	688.87ms	3	229.62ms	113.67ms	302.50ms	cudaMemcpy
18.63%	175.28ms	3	58.428ms	688.85us	173.90ms	cudaMalloc
7.60%	71.540ms	3	23.847ms	764.22us	35.400ms	cudaFree
0.41%	3.8528ms	364	10.584us	219ns	409.93us	cuDeviceGetAttribute
0.08%	763.07us	4	190.77us	165.27us	236.40us	cuDeviceTotalMem
0.04%	332.99us	4	83.246us	76.742us	91.516us	cuDeviceGetName
0.01%	84.781us	1	84.781us	84.781us	84.781us	cudaLaunch
0.00%	22.698us	1	22.698us	22.698us	22.698us	cudaDeviceSynchronize
0.00%	11.426us	5	2.2850us	156ns	10.061us	cudaSetupArgument
0.00%	8.2590us	12	688ns	236ns	2.9370us	cuDeviceGet
0.00%	7.6420us	1	7.6420us	7.6420us	7.6420us	cudaConfigureCall
0.00%	4.3420us	3	1.4470us	278ns	3.1340us	cuDeviceGetCount

nvprof trace mode

Where dct8x8 is the executable

```
$ nvprof --print-gpu-trace dct8x8

======== Profiling result:
     Start  Duration  Grid Size   Block Size   Regs  SSMem  DSMem  Size   Throughput  Name
  167.82ms  176.84us  -           -            -     -      -      1.05MB 5.93GB/s   [CUDA memcpy HtoA]
  168.00ms  708.51us  (64 64 1)   (8 8 1)      28    512B   0B     -      -          CUDAkernel1DCT(float*, …)
  168.95ms  708.51us  (64 64 1)   (8 8 1)      28    512B   0B     -      -          CUDAkernel1DCT(float*, …)
  169.74ms  708.26us  (64 64 1)   (8 8 1)      28    512B   0B     -      -          CUDAkernel1DCT(float*, …)
  170.53ms  707.89us  (64 64 1)   (8 8 1)      28    512B   0B     -      -          CUDAkernel1DCT(float*, …)
  171.32ms  708.12us  (64 64 1)   (8 8 1)      28    512B   0B     -      -          CUDAkernel1DCT(float*, …)
  172.11ms  708.05us  (64 64 1)   (8 8 1)      28    512B   0B     -      -          CUDAkernel1DCT(float*, …)
  172.89ms  708.38us  (64 64 1)   (8 8 1)      28    512B   0B     -      -          CUDAkernel1DCT(float*, …)
  173.68ms  708.31us  (64 64 1)   (8 8 1)      28    512B   0B     -      -          CUDAkernel1DCT(float*, …)
  174.47ms  708.15us  (64 64 1)   (8 8 1)      28    512B   0B     -      -          CUDAkernel1DCT(float*, …)
  175.26ms  707.95us  (64 64 1)   (8 8 1)      28    512B   0B     -      -          CUDAkernel1DCT(float*, …)
  176.05ms  173.87us  (64 64 1)   (8 8 1)      27    0B     0B     -      -          CUDAkernelQuantization (…)
  176.23ms   22.82us  -           -            -     -      -      1.05MB 45.96GB/s  [CUDA memcpy DtoA]
```


- **Regs**: Number of registers used per CUDA thread. This number includes registers used internally by the CUDA driver and/or tools.
- **SSMem**: Static shared memory allocated per CUDA block.
- **DSMem**: Dynamic shared memory allocated per CUDA block.

GPU-trace mode provides a timeline of all activities taking place on the GPU in chronological order.

Print individual kernel invocations and sort them in chronological order

Print CUDA runtime/driver API trace

```
$ nvprof --print-gpu-trace --print-api-trace dct8x8

======== Profiling result:
     Start  Duration  Grid Size  Block Size  Regs   SSMem   DSMem  Size   Throughput  Name
  167.82ms  176.84us  -          -           -      -       -      1.05MB 5.93GB/s  [CUDA memcpy HtoA]
  167.81ms    2.00us  -          -           -      -       -      -      -         cudaSetupArgument
  167.81ms   38.00us  -          -           -      -       -      -      -         cudaLaunch
  167.85ms    1.00ms  -          -           -      -       -      -      -         cudaDeviceSynchronize
  168.00ms  708.51us  (64 64 1)  (8 8 1)     28     512B    0B     -      -         CUDAkernel1DCT(float*, …)
  168.86ms    2.00us  -          -           -      -       -      -      -         cudaConfigureCall
  168.86ms    1.00us  -          -           -      -       -      -      -         cudaSetupArgument
  168.86ms    1.00us  -          -           -      -       -      -      -         cudaSetupArgument
  168.86ms    1.00us  -          -           -      -       -      -      -         cudaSetupArgument
  168.87ms       0ns  -          -           -      -       -      -      -         cudaSetupArgument
  168.87ms   24.00us  -          -           -      -       -      -      -         cudaLaunch
  168.89ms  761.00us  -          -           -      -       -      -      -         cudaDeviceSynchronize
  168.95ms  708.51us  (64 64 1)  (8 8 1)     28     512B    0B     -      -         CUDAkernel1DCT(float*, …)
```

nvprof --devices x --events y ./a.out

- x: device number (otherwise all devices will be profiled)
- y: event name (or comma separated events)
  - Gives very useful information, such as:
    - percentage of time at least one warp is active on a multiprocessor averaged over all multiprocessors on the GPU
    - achieved occupancy, ….

- You can also get all events**: --events all**
- Want to know all events: **nvprof –query-events**

nvprof --devices x --metrics y ./a.out

- x: device number (otherwise all devices will be profiled)
- y: event name (or comma separated events)
  - Gives very useful information, such as:
    - number of global memory loads, stores, …
    - number of global memory coalesced

- You can also get all events: **--metrics all**
- Want to know all events: **nvprof --query-metrics**

cuda-memcheck

- **memcheck** tool (default): capable of
  - detecting and attributing out of bounds and misaligned memory access errors in CUDA applications.
  - reporting hardware exceptions encountered by the GPU.
- **racecheck** tool: reports shared memory data access hazards that can cause data races.
- **initcheck** tool: reports cases where the GPU performs uninitialized accesses to global memory.
- **synccheck** tool: reports cases where the application is attempting invalid usages of synchronization primitives.

Example: cuda-memcheck --tool racecheck

cuda-memcheck

- **cuda-memcheck** [options] prog [args]
- Compile with –G –rdynamic –lineinfo
  - -G: forces the compiler to generate debug information for the CUDA application
  - -lineinfo: to generate line number information
  - -rdynamic: to retain function symbols

cuda-memcheck

- Example output:

```
========= Invalid __global__ write of size 4
=========     at 0x00000060 in memcheck_demo.cu:6:unaligned_kernel(void)
=========     by thread (0,0,0) in block (0,0,0)
=========     Address 0x400100001 is misaligned
```