Nolan Anderson | Week 2 Assignment 2 | CPE 613 | Janruary 21 2023

1. **Run the traditional Device Query from the CUDA samples. Show your output. What CUDA device did you run on?**
    - **Hint: Run "module load cuda" and, afterward, use this Makefile**

The CUDA device is a A100-SXM4-40GB.

```
CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 1 CUDA Capable device(s)

Device 0: "NVIDIA A100-SXM4-40GB"
  CUDA Driver Version / Runtime Version          11.7 / 11.7
  CUDA Capability Major/Minor version number:    8.0
  Total amount of global memory:                 40390 MBytes (42352050176 bytes)
  (108) Multiprocessors, (064) CUDA Cores/MP:    6912 CUDA Cores
  GPU Max Clock rate:                            1410 MHz (1.41 GHz)
  Memory Clock rate:                             1215 Mhz
  Memory Bus Width:                              5120-bit
  L2 Cache Size:                                 41943040 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(131072), 2D=(131072, 65536), 3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers  1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(32768, 32768), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total shared memory per multiprocessor:        167936 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 5 copy engine(s)
  Run time limit on kernels:                     No
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Enabled
  Device supports Unified Addressing (UVA):      Yes
  Device supports Managed Memory:                Yes
  Device supports Compute Preemption:            Yes
  Supports Cooperative Kernel Launch:            Yes
  Supports MultiDevice Co-op Kernel Launch:      Yes
  Device PCI Domain ID / Bus ID / location ID:   0 / 129 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 11.7, CUDA Runtime Version = 11.7, NumDevs = 1
Result = PASS
```

2. **Based on the specs in [this article,](#) and the relevant white papers, how does the NVIDIA H100 GPU with a PCIe Gen 5 board form-factor compare to the GV100 (Volta)?**

Specification	GV 100	H100
SM	80	114
TPC	40	57
FP32 Cores/SM	64	128
FP32 Cores/GPU	5120	14592
FP64 Cores/SM	32	128
FP64 Cores/GPU	2560	14592
Tensor Cores / SM	8	4
Tensor Cores / GPU	640	456
Boost Clock	1530MHz	1755MHz
Peak FP16 Compute (TFLOPs)	32.8	1600
Peak FP32 Compute (TFLOPs)	16.4	800
Peak FP64 Compute (TFLOPs)	8.2	48
Texture Units	320	456
Memory Interface	4096-bit HM2	5120-bit HBM2e
Memory Size	16GB	80GB
L2 Cache Size	6144KB	50MB
Shared Memory Size / SM	Up to 96KB	228KB
Register File Size / SM	256KB	2048KB
Register File Size / GPU	20480 KB	65536KB
TDP	300 Watts	300-350 Watts
Transistors	21.1 billion	80 billion
GPU Die Size	815 mm^2	814 mm^2
Manufactoring Process	12nm FFN	4nm

Accross the board, the H100 has significantly more compute than the GV100, and still has roughly the same TDP (thanks to the 4nm manufacturing process). Nearly 4x on the transistors, more than 4x the memory, higher boost clock, more FP32&64 cores etc. The table does a good job of comparing the PCIE implementations of both these generations. One thing to note is that the L2 cache size is significantly higher in the H100 (50MB vs 6144KB). Continually, the amount of shared memory and register file sizes in the SM's is much higher. This means we can store more data on the device and perform less memory accesses. This means the H100 is bound to perform better. Afterall, they're packing way more transistors into the same die size as a result of the 4nm manufacturing process. One major difference is that the H100 uses PCIe lanes so it is better at performing with x86 processors.

3. **What are the biggest differences in the SM from Ampere to Hopper (A100 to H100)?**

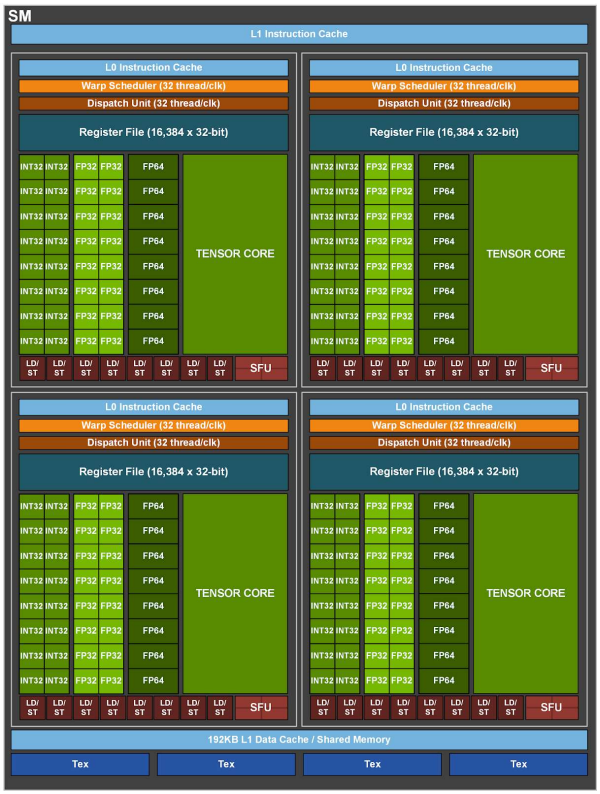
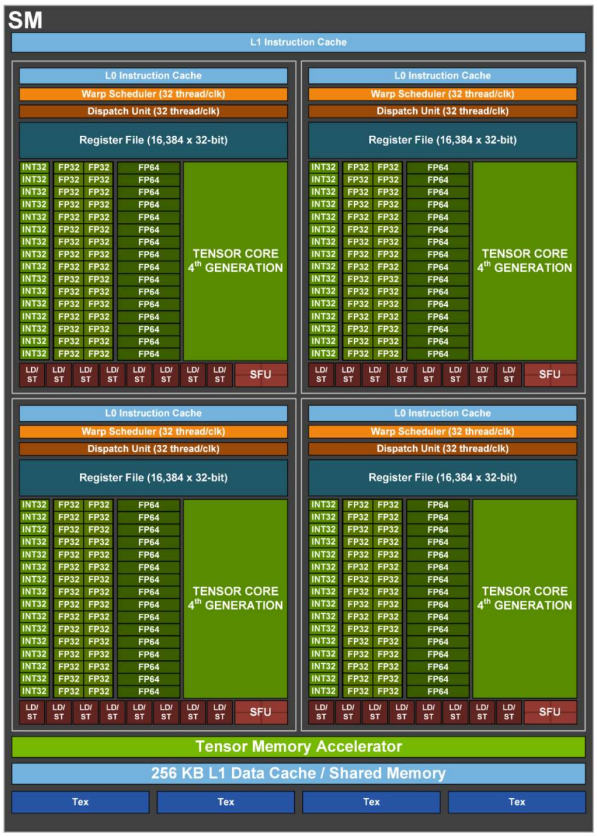
I am sourcing my information from this blog post by nvidia: https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/

One of the first things they mention is that the H100's major focus was to improve scaling for AI and HPC workloads, while becoming more efficient than the A100.

The main differences in the SM's is the new Fourth Generation tensor cores. There are more SM's and they run at higher clock speeds on the H100. Each H100 SM has double the Matrix Multiply-Accumulate performance of the A100. DPX Instructions accelerate dynamic programming algorithms by 7x, and 3x faster IEEE FP64/32 processing rates (2x faster clock-for-clock performance per SM). The thread block cluster allows for greater control of the threads on a single SM. The new SM's have distributed shared memory which allows direct communication between different SM's, which is pretty huge. Before the threads in a block could only communicate with each other. With the new H100 they

can all access shared memory. Though, it is likely slower than accessing L2 cache. Asynchronous execution allows for the Tensor Memory Accellorator to transfer large data efficiently between global and shared memory. This is the first time where a GPU has been asynchronous.

Next, I will compare the diagrams in the next table below. At the bottom, you can see the L1 Data cache is slightly larger, and also the addition of the tensor memory accelerator (as stated previously this speeds up global -> SM memory transfers). There is a reduction in the INT32 across the board (seems to be half). The warp scheduler, dispatch unit, instruction cache, register files and FP32/64's remain the same. The updated 4-gen tensor cores allow for higher performance. Overall, it seems like they increased **throughput** and **tensor core performance.** This is probably an over-generalization, but the previous paragraph explains how they do this in more detail.

A100	H100
	

4. **Explain the concept of a warp, how it is useful, and how it executes on the SM's hardware.**

SM’s are Streaming Multiprocessors and consist of multiple cores and shared memory and control. Threads and blocks require resources to execute, and we assign them to the SM’s. All threads in a block are assigned to a specific SM, and the remaining blocks wait for their turn. We can synchronize the threads in a block with __syncthreads() and they can access block specific memory. All threads in the same block are assigned to the same SM, as in you cannot split threads with different SM’s. Continually, you must assign a whole block to each SM. You cannot assign half a block to an SM, and threads from different blocks should not be synchronized. Therefore, because threads assigned to a SM run concurrently, we need a way to schedule them. This scheduler, called a warp, further divides these blocks that are assigned to an SM. Warp scheduling is important because you have a lot of cores that need to be filled up with threads (or work to do). While warps are device-specific, they are generally comprised to 32 threads. The threads in the warp are scheduled together and executed with the SIMD model. Basically, one instruction is loaded in an executed by all the cores, just with different data that is stored on the thread. Because you’re sharing the same instruction across all the different cores, the instruction fetch costs less.