Assignment 2:
GPU Architecture

Philipe Granhäll

November 24, 2022

https://github.com/granhall/DD2360HT22

Reflection on GPU-accelerated Computing

1. List the main differences between GPUs and CPUs in terms of architecture.

   (a) While CPUs today will almost always have a couple of multiprocessors when compared to a GPUs architecture, the difference is clear. GPUs have been designed with parallel processing in mind and thus come with a large number of multiprocessors.

   (b) CPUs are designed to efficiently handle any task thrown at them and thus the complexity of cores increases. This is a contrast to a GPU which makes use of more basic cores, often designed to be highly efficient at a specific mathematical computation.

   (c) CPUs are designed to provide low latency during operation, making use of certain techniques in cache and memory while a GPU is designed to have a high throughput. In basic terms, throughput indicates how quickly a new instruction can be executed after the previous one has finished.

2. Check the latest Top500 list that ranks the top 500 most powerful supercomputers in the world. In the top 10, how many supercomputers use GPUs? Report the name of the supercomputers and their GPU vendor (Nvidia, AMD, ...) and model.

   - Frontier - AMD - Instinct 250X
   - Fugaku - NO GPU
   - LUMI - AMD - Instinct 250X
   - Summit - NVIDIA - Tesla V100
   - Sierra - NVIDIA - Tesla V100
   - Sunway TaihuLight - NO GPU
   - Perlmutter - NVIDIA - A100
   - Selene - NVIDIA - A100
   - Tianhe-2A - NO GPU
   - Adastra - AMD - Instinct 250X

3. Check the latest Top500 list that ranks the top 500 most powerful supercomputers in the world. In the top 10, how many supercomputers use GPUs? Report the name of the supercomputers and their GPU vendor (Nvidia, AMD, ...) and model.

   - Frontier - $52 * 10^9$ flops/watt
   - Fugaku - $14 * 10^9$ flops/watt

- LUMI - $51 * 10^9$ flops/watt
- Summit - $14 * 10^9$ flops/watt
- Sierra - $12 * 10^9$ flops/watt
- Sunway TaihuLight - $6 * 10^9$ flops/watt
- Perlmutter - $27 * 10^9$ flops/watt
- Selene - $23 * 10^9$ flops/watt
- Tianhe-2A - 310^9 flops/watt
- Adastra - $49 * 10^9$ flops/watt

Device Query

Below we can find the output from running deviceQuery.

```
./deviceQuery Starting...

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

Detected 1 CUDA Capable device(s)

Device 0: "NVIDIA GeForce GTX 980"
  CUDA Driver Version / Runtime Version          11.8 / 11.8
  CUDA Capability Major/Minor version number:    5.2
  Total amount of global memory:                 4010 MBytes (4204920832 bytes)
  (016) Multiprocessors, (128) CUDA Cores/MP:    2048 CUDA Cores
  GPU Max Clock rate:                            1291 MHz (1.29 GHz)
  Memory Clock rate:                             3505 Mhz
  Memory Bus Width:                              256-bit
  L2 Cache Size:                                 2097152 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(65536), 2D=(65536, 65536), 3D=(4096, 4096, 4096)
  Maximum Layered 1D Texture Size, (num) layers  1D=(16384), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(16384, 16384), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total shared memory per multiprocessor:        98304 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 2 copy engine(s)
  Run time limit on kernels:                     Yes
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Disabled
  Device supports Unified Addressing (UVA):      Yes
  Device supports Managed Memory:                Yes
  Device supports Compute Preemption:            No
  Supports Cooperative Kernel Launch:            No
  Supports MultiDevice Co-op Kernel Launch:      No
  Device PCI Domain ID / Bus ID / location ID:   0 / 8 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 11.8, CUDA Runtime Version = 11.8, NumDevs = 1
Result = PASS
```

Figure 1: Bandwidth Test

Significant figures

- CUDA Capability: Tells us the feature available on the current card.
- CUDA Cores: Number of available CUDA cores
- Total amount of global memory: Data that can be stored on memory.
- GPU Max Clock rate: The clock cycles per second for a core, indicating how many instructions we can run per second.
- Bus width: Tells us the number of bits that can be transferred. Together with the memory clock rate, we can arrive at the memory bandwidth
- L2 Cache Size: A faster memory type for storing. A larger L2 cache can indicate better performance in certain scenarios.

Calculating GPU memory bandwidth using device query output:

$bandwidth = \text{bus width} * \text{memory clock rate} * \text{DDR}$

$bandwidth = 256\text{bits} * 1750\text{MHz} * 2$

$bandwidth = 224GB/s$

The memory bandwidth published by Nvidia for the GTX 980 is 224 GB/s so the calculation is consistent with what is published.

Compare GPU Architecture

Use the Internet to search to find the 3 latest NVIDIA GPU architectures in which you are interested.

- The 3 latest architectures by NVIDIA are the following
- Hopper
- Ampere
- Volta

Following is a table illustrating the change brought forth by these architectures

List the number of SMs, the number of cores per SM, the clock frequency and calculate their theoretical peak throughput

In this course I will be using my Nvidia GTX 980 which uses the Maxwell gen 2 architecture. Following is a table specifying its details.

Rodinia CUDA benchmarks and Profiling

Compilation of the selected benchmarks was a bit of a hassle. To begin, I am using Manjaro, an arch based distribution, where cuda and the cuda-toolkit on the AUR have a slightly differing structure to what seems to be the norm when downloaded off of Nvidia so I had to alter some destinations. Even then, it felt that something was off since some dependencies couldn't be found for some benchmarks but for others it was fine.

Table 1: 3 Latest NVIDIA Architectures

Architecture	Volta	Ampere	Hopper
Year	2017	2020	2022
Notable Features	- Tensor Cores - RT Cores - Second Gen NVLink	- Multi Instance GPU - Third Gen NVLink - PCIe Gen 4 Support - Third Gen Tensor Cores	- Forth Gen Tensor Core - PCIe Gen 5 Support - Forth Gen NVLink Support
Transistors	21.1 Billion	54.2 Billion	80 Billion
Memory Size	32 GB	40 GB	80 GB
L2 Cache Size	4 MB	40 MB	50 MB
Shared Memory Size	96 KB	164 KB	228 KB
TDP	300 Watts	400 Watts	350 Watts
TSMC Manufacturing Process	12nm	7nm	4nm

Table 2: Peak Throughput

Architecture	SM Cores	Cores per SM	Clock Frequency	Theoretical Peak Throughput
Volta	80	- FP32: 64 - FP64: 32 - INT32: 64 -Tensor: 8	1530 MHz	900 GB/s
Ampere	108	- FP32: 64 - FP64: 32 - INT32: 64 -Tensor: 4	1410 MHz	1555 GB/s
Hopper	114	- FP32: 128 - FP64: 64 - INT32: 64 -Tensor: 4	Not Listed	2000 GB/s (expected)

Table 3: Maxwell Architecture

Architecture	Maxwell
Year	2014
Notable Features	- Shared Memory Atomics update
Transistors	5.2 Billion
Memory Size	4 GB
L2 Cache Size	2 MB
Shared Memory Size	96 KB
TDP	165 Watts
TSMC Manufacturing Process	28nm
SM Cores	16
Cores per SM	- FP32: 128 - FP64: ? - INT32: ?
Clock Speed	1216 MHz
Theoretical Peak Throughput	224 GB/s

In the makefiles, I mainly changed paths to point to the correct CUDA folder and changed the compute capability specified in the makefiles to 5.2. For the OpenMP files, I did not have to make any changes.

I choose the benchmarks, Myocyte, LavaMD, and Particlefilter as I was able to compile the CUDA varients with small changes and they required less input than most of the others.

Parameters were chosen based on the suggestions in the README files within the application.

1. Myocyte Parameters
    - CUDA: 1000 1000 1
    - OpenMP: 1000 1000 1 4
2. LavaMD Parameters
    - CUDA: -boxes1d 10
    - OpenMP: -cores 4 -boxes1d 10
3. Particlefilter Parameters
    - CUDA: -x 1000 -y 1000 -z 100 -np 1000
    - OpenMP: -x 1000 -y 1000 -z 100 -np 1000

Following are the results I got for each.

Application	CUDA	OpenMP
Myocyte	9.86s	10.53s
LavaMD	0.17s	0.41s
Particlefilter	3.48s	2.83s

For the first two benchmarks, I got a speedup for running on the GPU. The Myocyte speedup is not very impressive but perhaps that depends on the problem that it is running. An interesting one to observe is the particlefilter. I believe that the issue here was that I was unable to specify threads to run on for the application so I believe it made full use of the 32 threads I have on my CPU. This was the same result I obtained when running it more than 20+ times.