Assignment II: GPU architecture

Franz Kaschner

08.11.2022

Exercise 1 - Reflection on GPU-accelerated Computing

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

CPUs	GPUs
Latency-oriented architecture: Minimize the running time of a single sequential program by avoiding task-level latency whenever possible.	Throughput-oriented architecture: Exploiting parallelism.
Excels at irregular control-intensive work.	Excels at regular math-intensive work with little synchronization.
Lots of hardware for control, fewer ALUs (tens of massive cores).	Thousands of small cores, but little hardware for control.

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.

https://www.top500.org/lists/top500/list/2022/06/

Rank	Name	GPU
1	Frontier	AMD Instinct MI250X
3	LUMI	AMD Instinct MI250X
4	Summit	NVIDIA Volta GV100
5	Sierra	NVIDIA Volta GV100
7	Perlmutter	NVIDIA A100 SXM4 40 GB
8	Selene	NVIDIA A100
10	Adastra	AMD Instinct MI250X

→ Seven supercomputers in the top 10 use GPUs.

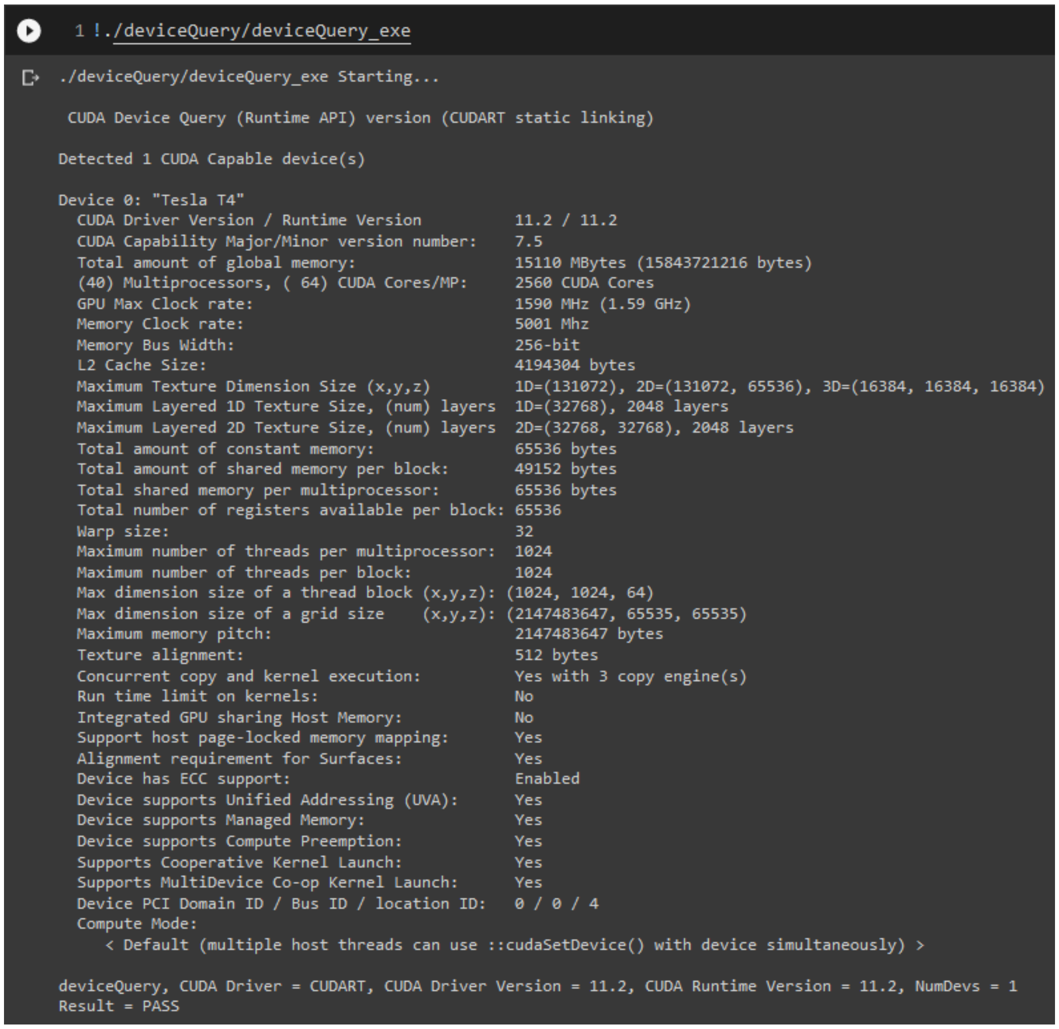
3. **One main advantage of GPU is its power efficiency, which can be quantified by Performance/Power, e.g., throughput as in FLOPS per watt power consumption. Calculate the power efficiency for the top 10 supercomputers. (Hint: use the table in the first lecture)**

https://www.top500.org/lists/top500/list/2022/06/
https://www.top500.org/lists/green500/list/2022/06/

Rank	Name	Energy Efficiency [GFLOPS/Watt]
1	Frontier	52.227
3	LUMI	51.629
4	Summit	14.719
5	Sierra	12.723
7	Perlmutter	27.374
8	Selene	23.983
10	Adastra	50.028

Exercise 2 - Device Query

1. **The screenshot of the output from you running deviceQuery test.**



```
!./deviceQuery/deviceQuery_exe
```

```
./deviceQuery/deviceQuery_exe Starting...

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

Detected 1 CUDA Capable device(s)

Device 0: "Tesla T4"
  CUDA Driver Version / Runtime Version          11.2 / 11.2
  CUDA Capability Major/Minor version number:    7.5
  Total amount of global memory:                 15110 MBytes (15843721216 bytes)
  (40) Multiprocessors, ( 64) CUDA Cores/MP:     2560 CUDA Cores
  GPU Max Clock rate:                            1590 MHz (1.59 GHz)
  Memory Clock rate:                             5001 Mhz
  Memory Bus Width:                              256-bit
  L2 Cache Size:                                 4194304 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:        65536 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  1024
  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 3 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 / 0 / 4
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 11.2, CUDA Runtime Version = 11.2, NumDevs = 1
Result = PASS
```

2. **What architectural specifications do you find interesting and critical for performance? Please provide a brief description.**

Especially important are:

- Number of SPs/CUDA cores: SPs are simple cores inside of SMs. Usually, a higher number of SPs indicates a higher performance.
- GPU Max Clock Rate: This value influences how many calculations per second one core can perform. Hence, a higher number indicates a higher performance.
- Memory Bandwidth (determined by memory bus width and memory clock rate): This value represents how fast data on the DRAM of the GPU can be accessed. In each memory clock cycle, the amounts of bits of the memory bus width (here: 256 bits) can be transferred (without considering DDR). Hence, a higher number indicates a higher performance.
- Total amount of global memory: Probably includes both the DRAM memory as well as the global L2 cache. A higher amount of memory is better as it allows to store more graphical or

computational data. For instance, more DRAM allows to use larger batch sizes when you train a CNN.
- L2 Cache Size: The L2 cache has a lower memory access time than the DRAM memory and more cache usually results in less cache misses. Hence, a higher number indicates a higher performance.

3. **How do you calculate the GPU memory bandwidth (in GB/s) using the output from deviceQuery? (Hint: memory bandwidth is typically determined by clock rate and bus width, and check what double date rate (DDR) may impact the bandwidth)**

$$\begin{aligned}\text{memory bandwidth} &= 2 \cdot \text{memory bus width} * \text{memory clock rate} = \\ &= 2 \cdot 256\,\text{bit} \cdot 5001\,MHz = 2.56 \cdot 1012\,\text{bit}/s = 320.1\,\text{GB}/s\end{aligned}$$

Factor 2 because of DDR.

4. **Compare your calculated GPU memory bandwidth with Nvidia published specification on that architecture. Are they consistent?**

On the of official website of the NVIDIA T4 the bandwidth is specified as “320+ GB/s”. In the datasheet the bandwidth is specified as “300 GB/s”.

Hence, the calculated GPU memory bandwidth is consistent with the published specification by Nvidia on that architecture.

Exercise 3 - Compare GPU Architecture

Three recent Nvidia GPU architectures: Volta, Ampere, Hopper
Source: https://www.nvidia.com/en-us/technologies/

1. **List 3 main changes in architecture (e.g., L2 cache size, cores, memory, notable new hardware features, etc.)**

Model	FP32	L2 cache size	Interconnect Bandwidth
V100 SXM2 (Volta)	15.7 TFLOPS	6144 KB	300 GB/s (NVLink)
A100 80GB SXM (Ampere)	19.5 TFLOPS	40MB	600GB/s (NVLink)
H100 SXM (Hopper)	67 TFLOPS	50MB	900GB/s (NVLink)

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

Sources:
- https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf
- https://developer.nvidia.com/blog/nvidia-ampere-architecture-in-depth/
- https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/
- https://www.quora.com/Can-I-find-out-how-many-TeraFlops-my-PC-can-calculate

Assumption: number of floating point operations per cycle = 1

Model	Number of SMs	Cores per SM	Clock frequency	Peak throughput (TFLOPS)
V100 SXM2 (Volta)	80	64 (FP32) 32 (FP64) 64 (INT32) 8 (Tensor)	1530 MHz	7.83 (FP32) 3.92 (FP64) 7.83 (INT32) 0.98 (Tensor)
A100 80GB SXM (Ampere)	108	64 (FP32) 32 (FP64) 64 (INT32) 4 (Tensor)	1410 MHz	7.83 (FP32) 3.92 (FP64) 7.83 (INT32) 0.49 (Tensor)
H100 SXM (Hopper)	132	128 (FP32) 64 (FP64) 64 (INT32) 4 (Tensor)	Not finalized	-

FLOPS = (number of SMs) * (cores per SM) * (number of clock cycles per second) * (number of floating-point operations per cycle).

3. Compare (1) and (2) with the NVIDIA GPU that you are using for the course.

I use the Nvidia Tesla T4 based on the Turing architecture.

Sources:
- https://arxiv.org/pdf/1903.07486
- https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-t4/t4-tensor-core-datasheet-951643.pdf
- https://www.tomshardware.com/news/nvidia-tesla-t4-turing-gpu,37788.html

Compare (1):

Model	FP32	L2 cache size	Interconnect Bandwidth
Nvidia Tesla T4	8.1 TFLOPS	4096 KB	32 GB/s (NVLink)

Compare (2):

Model	Number of SMs	Cores per SM	Clock frequency	Peak throughput (TFLOPS)
Nvidia Tesla T4	40	64 (CUDA cores) 8 (Tensor)	1590 MHz	4.07 (CUDA cores) 0.51 (Tensor)

Exercise 4 - Rodinia CUDA benchmarks and Profiling

I choose the following benchmarks:

- LavaMD2 (lavamd)
- LU Decomposition (lud)
- Myocyte (myocyte)

1. Compile both OMP and CUDA versions of your selected benchmarks. Do you need to make any changes in Makefile?

Required changes:

- LavaMD2 (CUDA) & LU Decomposition (CUDA): Set `-arch=sm_75` (because the Tesla T4 has the Turing architecture: https://askubuntu.com/questions/1157589/nvcc-fatal-value-sm-20-is-not-defined-for-option-gpu-architecture)
- LU Decomposition (OMP): Necessary changes in `./omp/Makefile.offload`:
  `CC := icc` → `CC := gcc`
  `CXX := icc` → `CXX := gcc`

No changes required in the other Makefiles.

2. Ensure the same input problem is used for OMP and CUDA versions. Report and compare their execution time.

Run commands:

- LavaMD2 (lavamd): `./lavaMD -cores 4 -boxes1d 10` (OMP), `./lavaMD -boxes1d 10` (CUDA)
- LU Decomposition (lud): `./omp/lud_omp -s 2048` (OMP), `./cuda/lud_cuda -s 2048` (CUDA)
- Myocyte (myocyte): `./myocyte.out 1000 100 1 4` (OMP), `./myocyte.out 1000 100 1` (CUDA)

Benchmark	OMP	CUDA
LavaMD2	6.530s	0.507s
LU Decomposition	0.740s	0.048s
Myocyte	12.010s	10.175s

3. Do you observe expected speedup on GPU compared to CPU? Why or Why not?

For all three benchmarks there is a speedup on GPU compared to CPU. With the LavaMD2 benchmark the total time is reduced by 92.2% and with the LU Decomposition by 93.5%. Hence, I would have expected that you also get similar values for the Myocyte benchmark but even with large inputs the total time there is only reduced by 15.3%.

(Bonus) Exercise 5 - GPU Architecture Limitations and New Development

Chosen paper:
Choukse, E., Sullivan, M. B., O'Connor, M., Erez, M., Pool, J., Nellans, D., & Keckler, S. W. (2020, May). Buddy compression: Enabling larger memory for deep learning and HPC workloads on GPUs. In 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA) (pp. 926-939). IEEE.

1. What limitations this paper proposes to address

High-throughput applications using GPUs need high memory bandwidths. However, memory with a high bandwidth usually only has a small capacity.
Hence, applications with a high memory footprint typically have to use workarounds (e.g., using multiple GPUs or explicitly managing the data transfer between GPU and CPU) which also have drawbacks.

2. What workloads/applications does it target?

It targets Deep Learning and HPC workloads.

3. What new architectural changes does it propose? Why it can address the targeted limitation?

Buddy Compression uses memory compression as a technique to expand the GPU memory. To be more specific, with Buddy Compression you assume that you can compress the data which you want to store in the memory of the GPU by a certain factor. For example, if you have 16GB of application data and a GPU with 8GB memory, you could use a compression factor of 2, so that all the data fits in the memory. Hence, you have to allocate 16GB/2 = 8GB of the memory.
However, it may happen that a memory-entry does not compress sufficiently. Then Buddy Compression enables that no re-allocations or page movements are required because Buddy Compression uses a slower but larger external memory (= buddy-memory) as an overflow storage. The technique still achieves a high performance because it uses high-bandwidth interconnects like NVLink, OpenCAPI or CXL with which the access time to the buddy-memory can be kept reasonably low, so that the total memory access time of the GPU is still high if most of the compressed data fits into the GPU memory. Fig. 1 (taken from paper) visualizes the basic principle behind Buddy Compression.
In fact, the evaluation has shown that Buddy Compression leads to a 1.5-1.9× GPU memory capacity when it was tested on a variety of different HPC and Deep Learning workloads, while the performance reduced only by 1-2%.

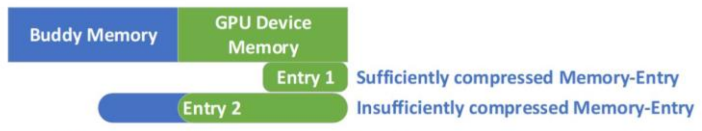

Buddy Memory
GPU Device Memory
Entry 1
Sufficiently compressed Memory-Entry
Entry 2
Insufficiently compressed Memory-Entry

Fig. 1: If a memory-entry (128B) does not compress sufficiently, part of it is accessed from the buddy-memory.

4. **What applications are evaluated and how did they setup the evolution environment (e.g., simulators, real hardware, etc)?**

As HPC workloads the authors use a subset of SpecACCEL OpenACC v1.2 and CUDA versions of the DOE benchmarks HPGMG and LULESH.
As DL workloads the authors train a set of 5 CNNs (AlexNet, Inception v2, SqueezeNetv1.1, VGG16, and ResNet50) implemented using the Caffe framework. For training they use the ImageNet dataset. Furthermore, they train a LSTM network.
As evaluation environment the authors use a dependency-driven GPU performance simulator. The simulator configurations are based on public information from the NVIDIA P100 Pascal GPU and the interconnect characteristics of recent Volta GPUs and non-public architectural information are taken from microbenchmark results.

5. **Do you have any doubts or comments on their proposed solutions?**

The proposed technique provides a good solution to expand the GPU memory. However, if high-bandwidth interconnects like NVLink are not available and only PCIe can be used, then this method would perform much worse.